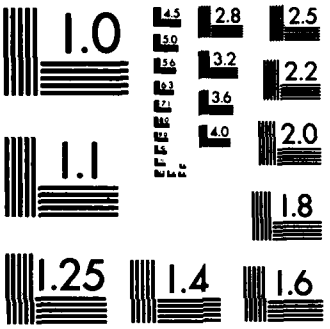
AD-A135 912 UCLA DEMODULATION ENGINE(U) CALIFORNIA UNIV LOS ANGELES 1/2
DEPT OF COMPUTER SCIENCE R SADR MAR 83 UCLA-ENG-83-23
MDA903-82-C-0064

UNCLASSIFIED F/G 9/5 NL



UCLA

COMPUTER SCIENCE DEPARTMENT

UCLA DEMODULATION ENGINE

Ramin Sadr

May 1983
. CSD-830518
ENG-8323

COMPUTER SCIENCE DEPARTMENT OFFICERS

Dr. Algirdas Avižienis, Chairman
Dr. Bertram Bussell, Vice Chairman
Dr. Milos Ercegovac, Vice Chairman
Dr. Gerald Popek, Vice Chairman
Mrs. Arlene C. Weber, Management Services Officer

CENTER FOR EXPERIMENTAL COMPUTER SCIENCE

Dr. Gerald Popek, Director
Dr. Terence Gray, Associate Director

MANUFACTURING ENGINEERING PROGRAM

Dr. Michel A. Melkanoff, Director

This report is part of a continuing series of technical reports initiated in January 1981 and is issued by the Computer Science Department at UCLA. This technical report presents the latest research results established by faculty members and research staff of the Department. UCLA Computer Science Department Technical Reports are directed to the professional community and range from the presentation of short technical contributions to complete Ph.D. dissertations. For a complete list of reports in this series you may contact the Computer Science Department Archivist at the address below.

University of California, Los Angeles
Computer Science Department
School of Engineering and Applied Science
3732 Boelter Hall
Los Angeles, California 90024

Computer Science Report No. CSD-83-1
UCLA Report No. ENG-83-23
March 1983

UCLA Demodulation Engine

by
Ramin Sadr

Computer Science Department
School of Engineering and Applied Science
University of California
Los Angeles

This research, conducted under the direction of Professors Vance Tyree and Jim K. Omura, was sponsored by the Defense Advanced Research Projects Agency, Department of Defense Contract MDA 903-82-C-0064 (Advanced Teleprocessing Systems).

ABSTRACT

This report describes the VLSI design and implementation of a Viterbi algorithm processor for simultaneous data demodulation and phase tracking of Minimum Shift Keying signal.

During the 1981-82 academic year, graduate students in the VLSI course (CS258A-C) at UCLA designed the implementation of this system as a one-year class project, and with support from ARPA (Advanced Research Project Agency of the Department of Defense), fabricated this processor on a single chip, using 4-micron NMOS technology. *UCLA Demodulation Engine* can be used as an inexpensive digital radio receiver in a variety of applications.

Background and Acknowledgment

In August 1981, Professor Jim Omura submitted a proposal to ARPA for support of my research to design a VLSI chip for a Viterbi algorithm processor that simultaneously performs data demodulation and phase tracking of Minimum Shift Keying signals.

Under supervision of my Ph.D. advisor, Dr. Jim Omura, the VLSI system architecture was developed in September 1981. I sincerely thank him for his guidance and encouragement throughout the project.

In November 1981, Vance Tyree, the coordinator and instructor of the VLSI program at UCLA, decided to adopt the implementation of this Viterbi algorithm processor as a class project for the UCLA one-year graduate course sequence on VLSI. Professor Tyree was instrumental in the fabrication of the chip and supervision of related design activities. He supervised my work while I was also involved in coordinating student project groups in the design effort.

I acknowledge our debt to ARPA for funding this project; to ISI (Information Science Institute) for handling fabrications and making other services available for the project; and to UC Berkeley for providing the Caesar software package, which definitely made the effort in the implementation phase of this project very time effective. I would also like to thank Lillian Larijani and Terry Peters for editing and assembling the final report and Ruth Pordy for the illustrations.

Contributions made to the project by other participating graduate students are acknowledged at the beginning of each relevant chapter. I must also make special mention of the interest and accommodations provided by Dr. Leonard Kleinrock of the Computer Science Department.

Table of Contents

List of Figures

page

CHAPTER 1
Formulation and Description of UCLA Demodulation Engine

1.1 Introduction

This report describes the design and implementation of a chip for simultaneous data demodulation and phase tracking of Minimum Shift-Keying (MSK) signal using the Viterbi algorithm. Knowledge of the basic Viterbi algorithm is assumed throughout this report.

MSK belongs to a larger class of modulations called Continuous Phase tulations (CPM) which, because of their superior spectral characteristics, are also referred to as width efficient modulation techniques [6,7,8]. The need for efficient use of bandwidth has grow nsiderably with increased usage of digital techniques to transfer data, voice, facsimile, video inform ic.

The practical application of these modulations has been limited by a lack of an effective phase estimation algorithm. With simultaneous phase estimation and data detection, the Viterbi algorithm overcomes this fundamental problem and eliminates the need for a separate phase tracking system.

Using the generalized Viterbi algorithm [3,4] to find the optimum sequence is equivalent to the dynamic programming solution for estimating the states of a finite state machine. The Viterbi algorithm has many other applications in the areas of convolutional codes, intersymbol interference channel, data compression, text recognition, etc. [1,4]. It is hoped to present such a general framework for design of the Viterbi algorithm processor that the architecture of this VLSI chip could be easily modified for other applications.

1.2 Final Chip

A typical role of this chip in a digital radio is shown in Figure 1.1

This chip is fabricated using 4 micron* NMOS technology. The design methodology and techniques used are based on the Mead & Conway [2] approach to VLSI system design. This system, packaged on a single 64-pin package, occupies a full size 7150x7100 micron^2 die. Operating at 15 MHz, it has an effective bit rate of about 650 Kbps. This chip has 1000 bits of memory organized as 30 bit long First-In First-Out (FIFO) registers with special background/foreground data transfer capability and fast 6-bit binary multiplier in the signed magnitude form to compute two dimensional vector inner product for real numbers. Testing is facilitated by using level sensitive scan design techniques in the controller

*λ=2 *micron*, where minimum feature size is 2λ.

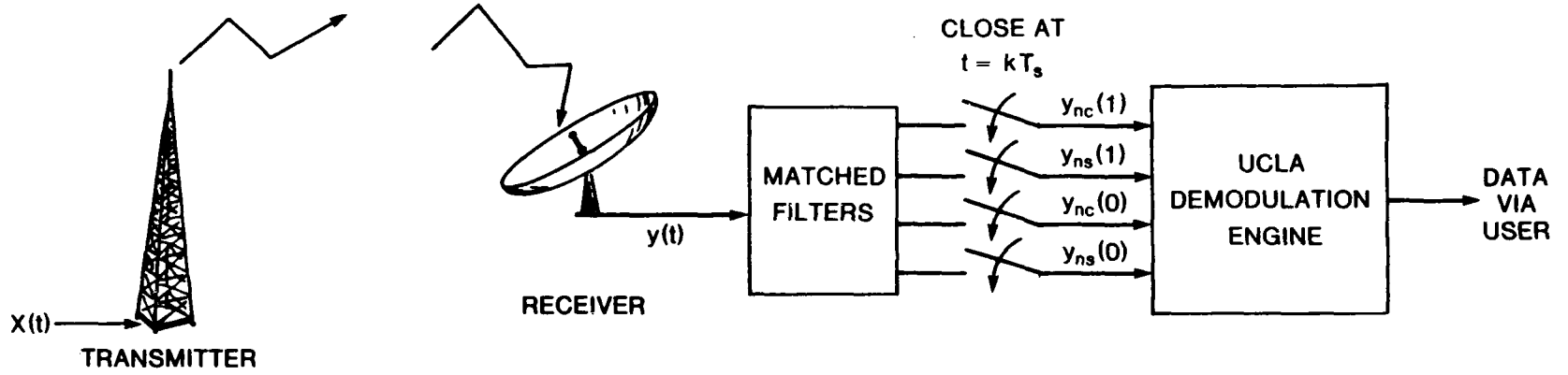


Figure 1.1. UCLA Demodulation Engine

section and by direct links to the inner cells of the system via the output pads. There is a global Reset on the chip to interrupt the system operation and reset all the sub-systems. All cells and subcells are custom designed for this chip.

The design is a synchronous sequentially pipelined Viterbi Algorithm processor. It assumes bit timing is obtained external to the chip.

1.4 MSK

The complete derivation of the demodulator for simultaneous data demodulation and phase tracking of the MSK signal is contained in the appendix A. The present section is a summary of those results in the appendix which will be used in Chapter 2.

The MSK transmitted signal has the form

$$x(t) = \sqrt{2P}\ Cos(\omega_c t + \theta(t))$$

where for MSK

$$\theta(t) = a_n \frac{\pi}{2}\left(\frac{t-nT_s}{T_s}\right) + \sum_{i=-\infty}^{n-1} \frac{\pi}{2} a_i$$

$$nT_s \leqslant t < nT_s$$

and

ω_c = Carrier frequency
T_s = Symbol duration time
a_i = Transmitted data where $a_i \in \{+1,-1\}$ *
E_s = Energy of the transmitted signal
$P = \frac{E_s}{T_s}$

For the usual point-to-point communication channel, the signal at the receiver is

$$y(t) = x(t,\theta) + n(t)$$

where n(t) is white Gaussian noise with double-sided flat spectral density $S_n(\omega) = N_o/2$.

With ideal knowledge of phase and frequency, the optimum MSK receiver computes

$$y_{nc}(1) = \int_{nT_s}^{(n+1)T_s} y(t)\sqrt{2P}\ Cos(\omega_c t + \frac{\pi}{2}(t-nT_s)/T_s)\,dt$$

* The bit "1" corresponds to +1 and bit "0" to -1.

$$y_{ns}(1) = -\int_{nT_s}^{(n+1)T_s} y(t)\sqrt{2P}\ Sin(\omega_c t+\frac{\pi}{2}(t-nT_s)/T_s)\,dt$$

$$y_{nc}(0) = \int_{nT_s}^{(n+1)T_s} y(t)\sqrt{2P}\ Cos(\omega_c t-\frac{\pi}{2}(t-nT_s)/T_s)\,dt$$

$$y_{ns}(0) = -\int_{nT_s}^{(n+1)T_s} y(t)\sqrt{2P}\ Sin(\omega_c t-\frac{\pi}{2}(t-nT_s)/T_s)\,dt$$

and uses the metric

$$bm(a_n) = m(\underline{y}_n;S_n,a_n) = y_{nc}(a_n)Cos\,S_n + y_{ns}(a_n)Sin\,S_n$$

$$\text{where } \underline{y}_n = (y_{nc}(1), y_{nc}(0), , y_{ns}(1), y_{ns}(0))$$

in a four state Viterbi decoder. Here $S_n = \sum_{i=-\infty}^{n} \frac{\pi}{2} a_i$ taking values in $\Phi=\{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\}$.

The bit error probability curve for coherent MSK is shown in Figure 1.2.

With unknown values of carrier phase, the phase space $[0,2\pi)$ is quantized into Q equal spaced intervals. The unit circle is approximated by the quantized phase space $\Phi = \{0, \Delta, 2\Delta, \ldots, (Q-1)\Delta\}$ where $\Delta = \frac{2\pi}{Q}$. Random phase perturbations are modeled as equally likely transitions to the adjacent quantized phase states; hence the state transition equation becomes

$$S_n = S_{n-1} + a_{n-1}\frac{\pi}{2} + \phi_{n-1}$$

where the discrete random phase perturbations $\phi_n \in \{-\Delta,0,+\Delta\}$ and $a_n \in \{-1,+1\}$ for all n. The branch metric expression in this case is

$$m(S_n;S_{n+1}) = y_{nc}(a_n)Cos(S_n+\phi_n) + y_{ns}(a_n)Cos(S_n+\phi_n) \qquad (1.1)$$

We shall denote $m(S_n;S_{n+1})$ as $bm(a_n)$ for notational simplicity.

The bit error probability bound for this demodulator as a function of signal-to-noise ratio $\left(E_s/N_o\right)_{dB}$ is shown in Figure 1.3 for ideal known phase case and for both 16 and 32 levels of quantization (Q=16,32) with unknown phase perturbations. The bound itself is about 1 db off from the ideal case for known constant phase. Additional degradation due to the random phase term is small even for 16 point quantization of the phase space.

We assume throughout this work that bit timing is ideal and available from outside the Viterbi algorithm processor.

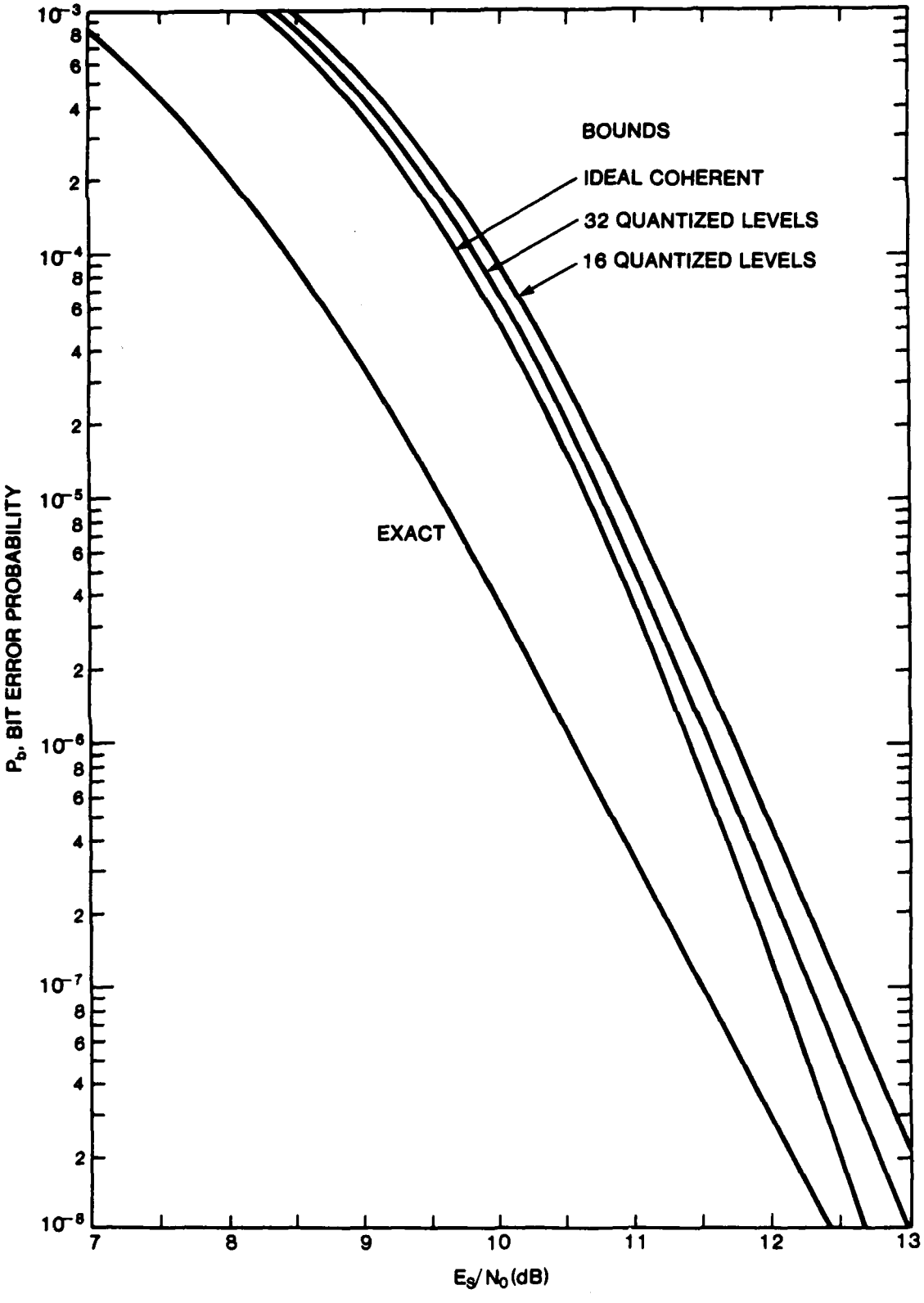


Figure 1.2. Bit Error Probability for the MSK using the Viterbi Algorithm

CHAPTER 2
VLSI System Design

2.1 Introduction

The system design in this chapter describes every subsystem in terms of:

a. inputs and outputs.

b. digital units: registers, counters, arithmetic logic unit (ALU), programmable logic array (PLA).

c. microprograms: programs consisting of microsequences which chronologically describe data transfer within the system (included in the Appendix B).

The design of the Viterbi algorithm processor is specialized in this chapter for simultaneous data demodulation and phase tracking of the MSK signal with random phase.

A single processor architecture is considered where there is one central processing unit (CPU) which results in a sequential form of the Viterbi algorithm. This assumption significantly simplifies the implementation task and, upon successful completion, could easily be generalized to a parallel multiprocessor design for higher data rate. The generalized design approach is further discussed in Section 2.8.

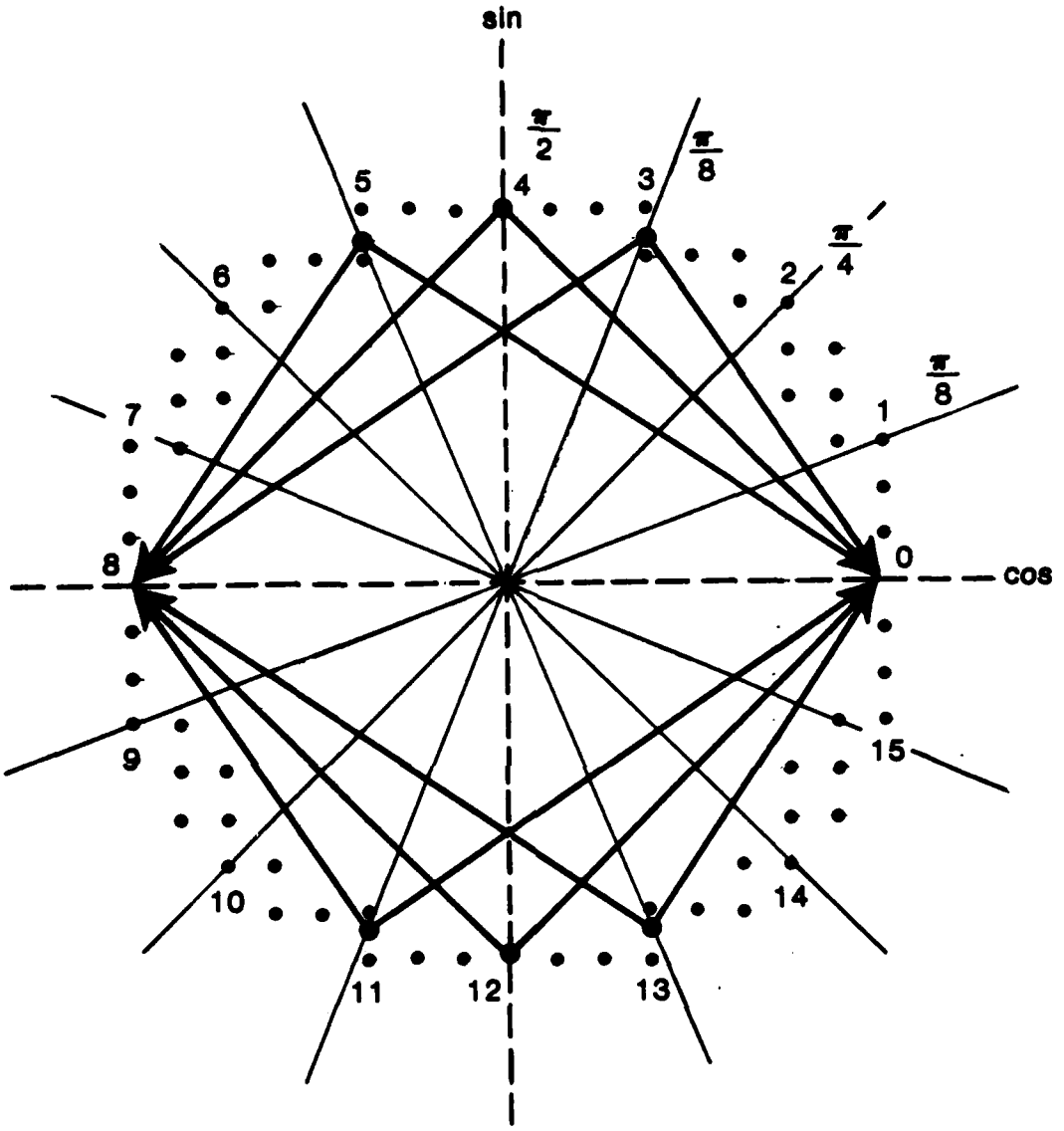
The system design of the Viterbi algorithm processor is closely related to simulating the structure of the trellis diagram . Knowledge of all possible transitions during one period of the trellis diagram in our case of study will uniquely determine the trellis diagram for all time $t = n\ T_s$, $n = 1,2,3,...$

2.2 Overview

The overview section outlines the method for obtaining the trellis diagram in our case and a brief review of the Viterbi algorithm.

Quantizing the phase space $(0,2\pi]$ around the unit circle into 16 equal space intervals gives quantized phases or "states" as shown in Figure 2.1. The phase space is then

$$\Phi = \{0, \frac{\pi}{8}, \frac{2\pi}{8}, \frac{3\pi}{8}, \ldots, \frac{15\pi}{8}\}$$



M.S.K. STATE TRANSITION EQ:

$$S_n = S_{n-1} + a_{n-1}\frac{\pi}{2} + \phi_{n-1}$$

$$\phi_n \in \{0, +\frac{\pi}{8}, -\frac{\pi}{8}\} \quad a_n \in \{+1, -1\}$$

Figure 2.1. Phase Space

The state transition equation for the MSK with random phase is

$$S_n = S_{n-1} + a_{n-1}\frac{\pi}{2} + \phi_n$$

where $S_n \in \Phi$, $a_n \in \{+1,-1\}$ and $\phi_n \in \{\frac{\pi}{8},-\frac{\pi}{8},0\}$.

In order to find all possible transitions defined on the phase space, note that, for MSK, when "bit 1" is transmitted, it causes a rotation of $+\frac{\pi}{2}$; when "bit 0" is transmitted, it causes a rotation of $-\frac{\pi}{2}$. Phase drifts are modeled as equally-likely transitions to adjacent phase values of the corresponding state as shown in Figure 2.1 for State #1. Note that there are six possible transitions to each state,3 for bit "1" ,3 for bit "0".

The numbering of the states in the trellis diagram is arbitrary; numbers are assigned to the sixteen states counter-clockwise around the unit circle as shown in Figure 2.1.

The transition Table 1 summarizes all the useful information represented by the trellis diagram in one transition. In this table the first column is the present state, and the next six columns are those last states which lead to the present state.

The notion of the present states and the last states of the trellis diagram will be used often. The "Present States" denoted as I are the states of the the trellis diagram at time nT_s, and the "Last States" denoted as LS are the states at time $(n-1)T_s$, as shown in Figure 2.8.

Present State	Last States						X1	X2
I	LS1	LS2	LS3	LS4	LS5	LS6		
1	4	5	6	12	13	14	1	0
2	5	6	7	13	14	15	b	a
3	6	7	8	14	15	16	c	c
4	7	8	9	15	16	1	a	b
5	8	9	10	16	1	2	0	1
6	9	10	11	1	2	3	-a	b
7	10	11	12	2	3	4	-c	c
8	11	12	13	3	4	5	-b	a
9	12	13	14	4	5	6	-1	0
10	13	14	15	5	6	7	-b	-a
11	14	15	16	6	7	8	-c	-c
12	15	16	1	7	8	9	-a	-b
13	16	1	2	8	9	10	0	-1
14	1	2	3	9	10	11	a	-b
15	2	3	4	10	11	12	c	-c
16	3	4	5	11	12	13	b	-a
Phase:	$-\pi/8$	0	$\pi/8$	$-\pi/8$	0	$\pi/8$		
	Transition caused by "bit 0"			Transition caused by "bit 1"				

a=Sin $\pi/8$=.383, b=Cos $\pi/8$=.924, c=Cos $\pi/4$=.707

Table 1. Transition Table

The trellis diagram is obtained by evolving all the possible transitions in time as shown in Figure 2.2.a In this case of study, the trellis diagram is composed of too many transitions; therefore, only a portion of the trellis diagram is depicted. The key point here is that all possible sequences of phase and data are represented by paths in the trellis diagram. The most likely path is found by the Viterbi algorithm.

The Viterbi algorithm is summarized in Figure 2.2.b. This flowchart is a simple representation of how this algorithm is used to find the optimum path in the trellis diagram. This flowchart outlines the sequencing of the Viterbi algorithm on the the trellis diagram; in this case the trellis diagram is composed of 16 states, and a single processor is used to find the survivor at each state. Here

Acc met(i;N) = Accumulated metric of the state i at time N
m(i,j) = branch metric value for transition j-->i (j leads to i).

The branch metric values are defined only for the subset of the last states which are connected to the present state i on the trellis diagram; the branch metric values for each present state are determined by the observed signal y(t) and (X1,X2). From 1.1 the branch metric values are

$$m(S_n;S_{n+1}) = y_{nc}(a_n)Cos(S_n+\phi_n)+y_{ns}Sin(S_n+\phi_n)$$

Let $X1=Cos(S_n+\phi_n)$ and $X2=Sin(S_n+\phi_n)$. The estimated phase values for the transitions caused by bit "1" to the present statel for LS1 is $-\frac{\pi}{8}$, for LS2 is 0 and for LS3 is $+\frac{\pi}{8}$. Hence, for example X1 in each case is

for $LS1\ X1=Cos(LS2+\frac{\pi}{8}-\frac{\pi}{8})$,for $LS2\ X1=Cos(LS2+0)$,for $LS3\ X1=Cos(LS2-\frac{\pi}{8}+\frac{\pi}{8})$.

Therefore, (X1=Cos(LS2),X2=Sin(LS2)), similarly for bit"0" the above is true with LS2 replaced by LS5.

$$m(i,j) = \begin{cases} bm(1)= y_c(1)\,X1+y_s(1)\,X2 & bit\ -1-\ (X1=Cos(LS2),X2=Sin(LS2)) \\ bm(0)= y_c(0)\,X1+y_s(0)\,X2 & bit\ -0-\ (X1=Cos(LS5),X2=Sin(LS5)) \end{cases}$$

These branch metric values are real valued. Accumulated metric values of all sixteen states are initially set at zero. A new set of m(i,j) values is available every T_s second, during which time it is necessary to to find the survivors of all sixteen present states of the trellis diagram. The branch chosen at each state is such that it maximizes the accumulated metric value of the present state; this branch is the so-called "survivor." The bit which causes this transition is stored. This results in 16 survivors at each state during each T_s second. Applying this procedure for nT_s n = 1,2,3,4.... seconds, the survivors comprise 16 connected paths. While applying the Viterbi algorithm to find the optimal path, the

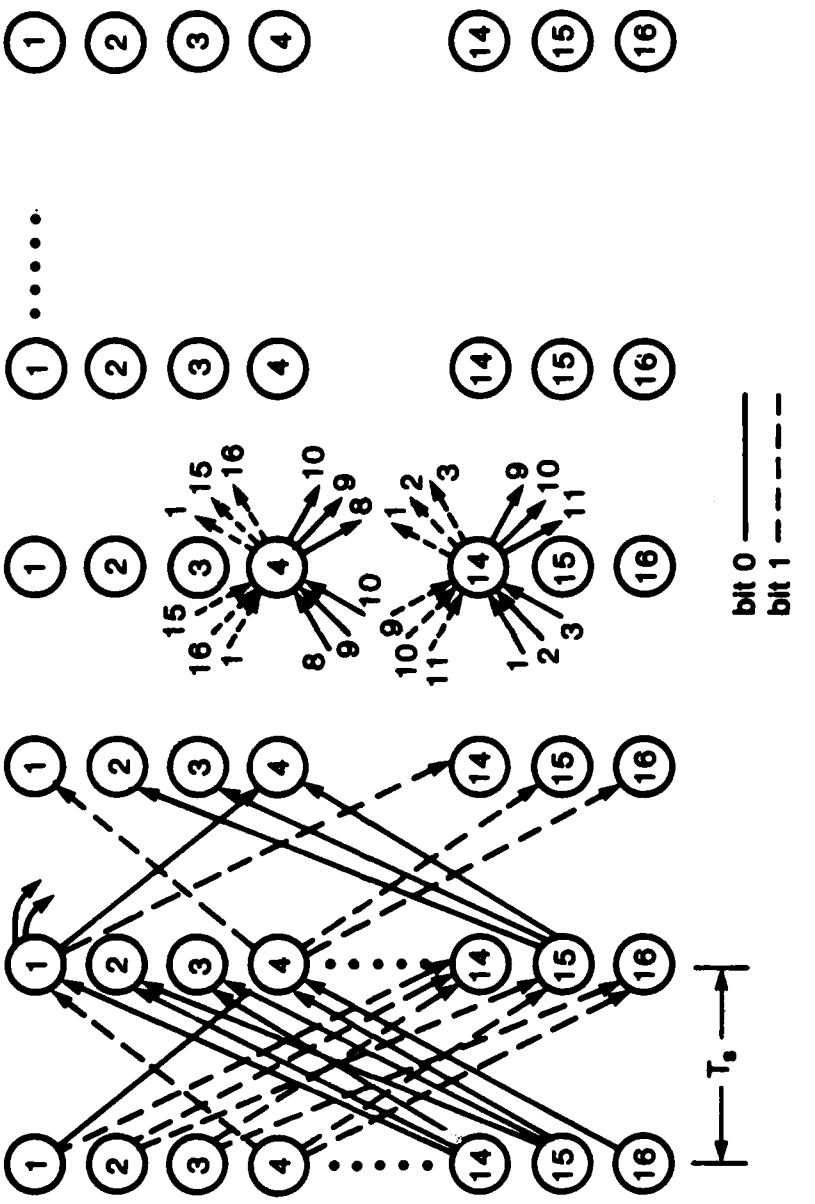


Figure 2.2a. Trellis Diagram

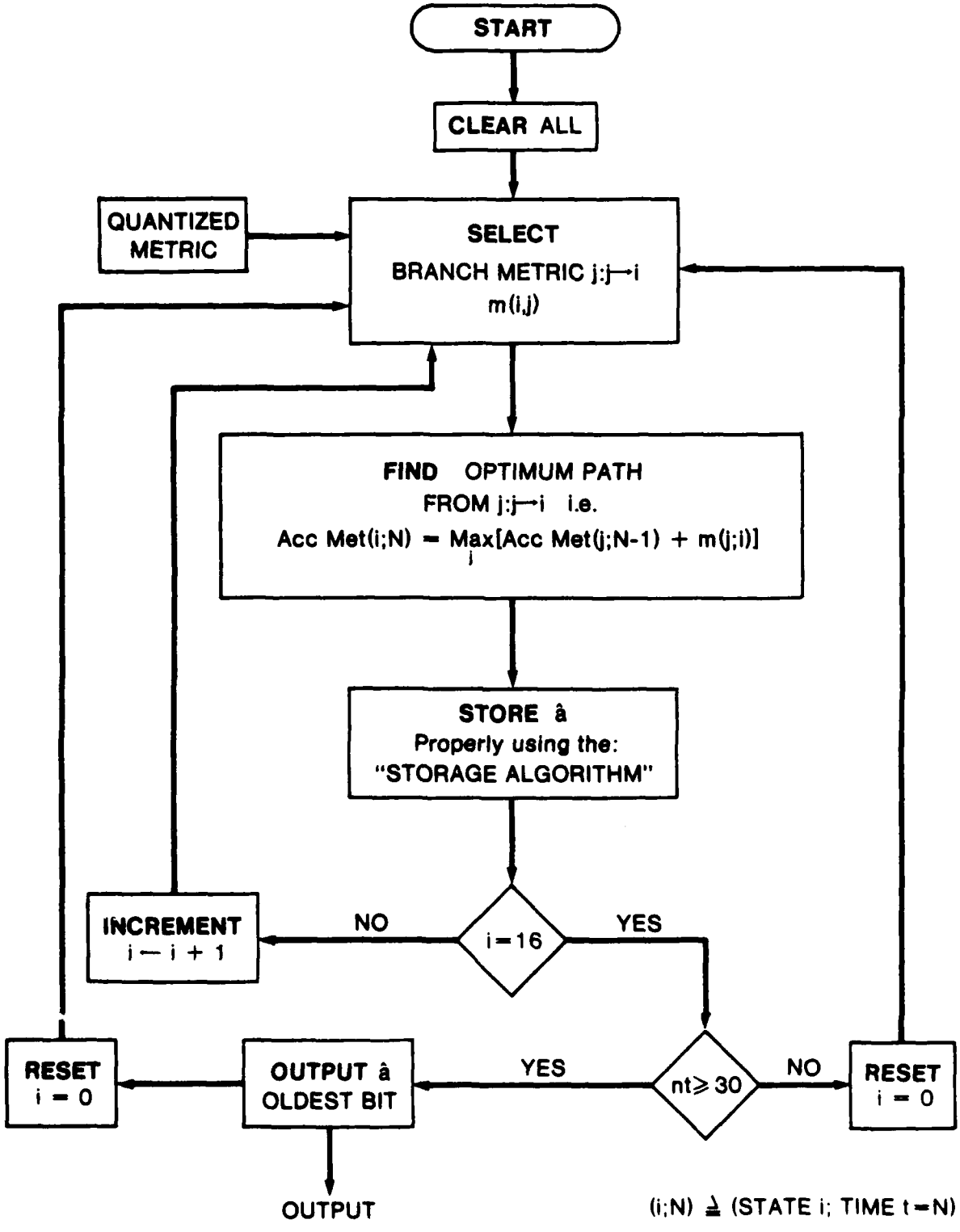


Figure 2.2b. Viterbi Algorithm — Flow Chart

tails of all 16 paths, merge together (with high probability) after a sufficiently long time. The merged tail is indeed the optimal path. In our case, after $30T_s$, we start outputting the oldest bit. Truncation of the path memory is discussed further when we describe the memory. A typical situation at $t=30T_s$ is shown in Figure 2.3.

The following terminology is defined for cycles of the trellis diagram which will be used in this chapter (Refer to Figure 2.2).

i. Symbol cycle: epoch taken to perform 16 nodal cycle, this time is equal to the data duration time Ts.

ii. Nodal Cycle: epoch taken to find the survivor of a state. This period consists of 6 branch cycles.

iii. Branch Cycle: epoch taken to add the branch metric to the last state of the accumulated metric value.

The system design begins at the inputs to this chip. It will be shown that these inputs are the quantized In-phase and Quadrature components of the observed signal y(t) during each Ts second. The quantization is necessary to perform digital processing.

2.3 Branch Metric Generator

It is not possible to use m(i,j) as the inputs to this chip for all possible values of i and j. The equations for m(i,j) are restated here

$$y_c(1)X1+y_s(1)X2 \quad \text{"1"}$$
$$y_c(0)X1+y_s(0)X2 \quad \text{"0"}$$

where $y_c(1)$, $y_s(1)$ $y_c(0)$ and $y_s(0)$ are the In-phase and Quadrature components of the observed signal for bit "1" and "0".

It can be deduced from the above equation or table 1 that there are 32 unique values of m(i,j), and if 4 bit quantization is assumed, this will result in 128 inputs to the system. This point is an example of system design trade-off. For practical implementation, this number of input pins is physically unrealistic on a single package.

The solution adopted is shown in Figure 2.4[1].

[1]Due to round-off error, this solution is less accurate than having the quantized m(i,j) available.

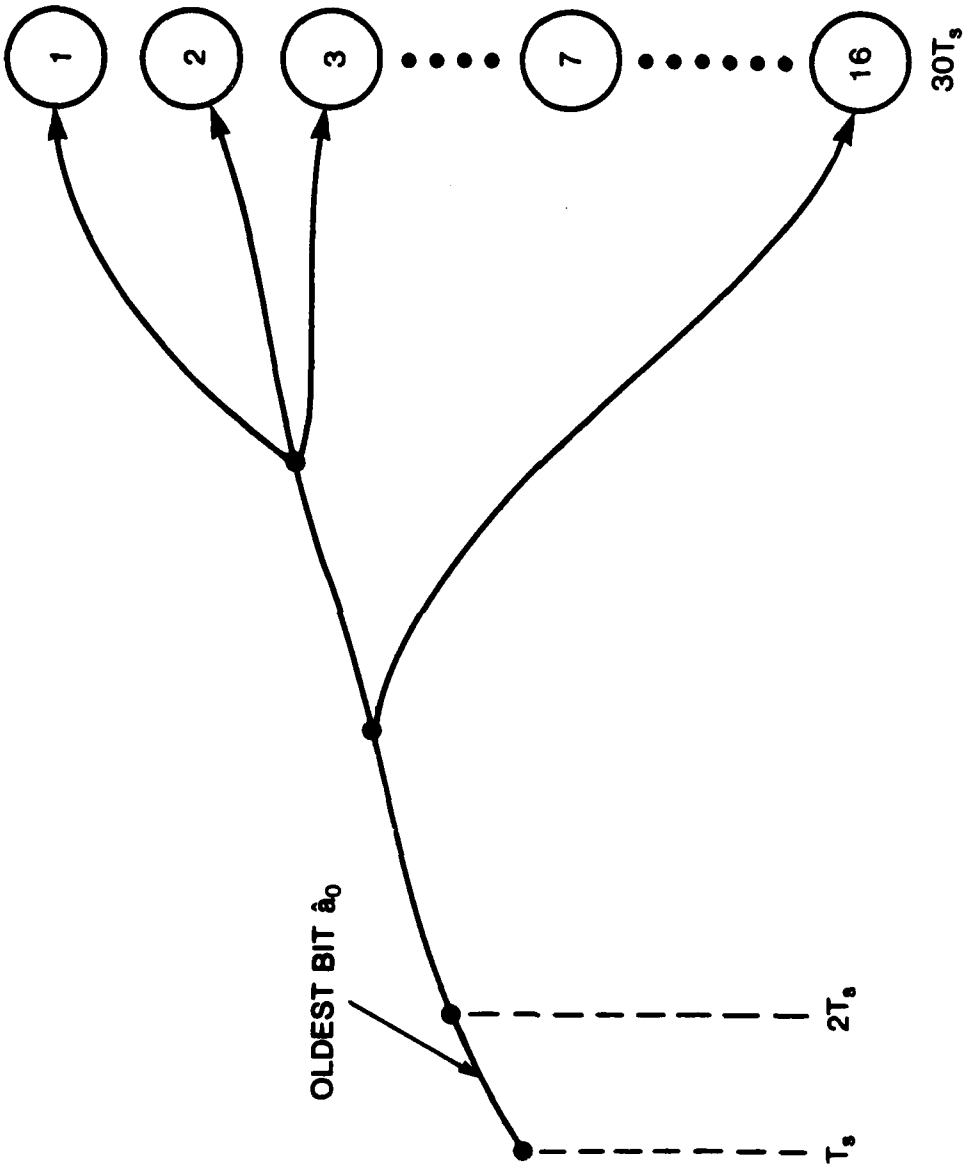


Figure 2.3. Path Merging

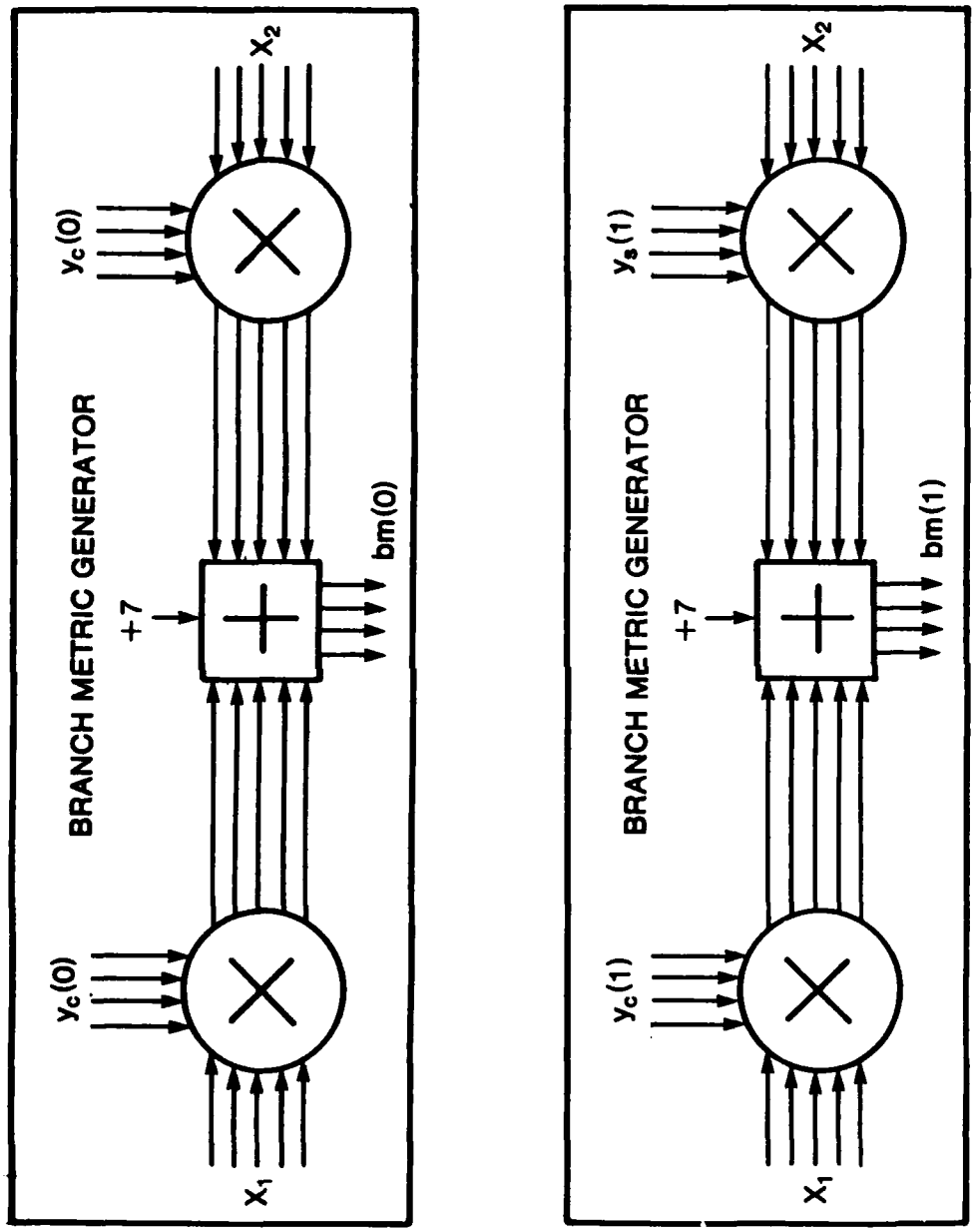


Figure 2.4. Multiplier

This solution introduces two multiplications and one addition operation to the complexity of the system requirement. Its advantage is that it requires only 16 input pins consisting of 4 bits each for $y_c(1)$, $y_s(1)$ $y_c(0)$ and $y_s(0)$ where in binary signed magnitude form taking values in [-7,7] interval. These inputs are quantized outside the chip using a 15-level quantizer with input-output relationship as shown in Figure 2.5.

Every state of the trellis diagram possesses a unique vector (X1,X2); the inner product of this vector is taken by the appropriate In-phase and Quadrature components input to generate the branch metric values. X1 and X2 take values in the [-1,1] interval. Each of these two inputs is represented by six bits in fixed point signed magnitude form. A factor of +7 is also added to all the branch metric values inside the Multiplier, to make the branch metric values non-negative integers to simplify comparison tasks by the CPU (discussed in section 2.5.2) .

2.4 Trellis Processing Unit

The function of this subsystem is to

i. model the connectivity of the states of the trellis diagram;

ii. generate X1, X2.

Each of the sixteen present state #s and sixteen last state #s is coded using 4 bits in binary presentation form.

The block diagram for the TPU is shown in Figure 2.7. The 4-bit I-counter points to the present state I and is incremented at the end of a nodal cycle.

The TPU is designed to output all last states connected to the present state I on the trellis diagram, i.e., for a given present state, it outputs the row of the last states shown in Table 1. This is accomplished simply by noting the following relations between the present states and the last state using modulo-15 addition. These relations become obvious by referring to Table(1).

$$LS1 = I \oplus 3,\ LS2 = LS1 \oplus 1,\ LS3 = LS2 \oplus 1$$
$$LS4 = I \oplus 11,\ LS5 = LS4 \oplus 1,\ LS6 = LS5 \oplus 1$$
$$\oplus \equiv \textit{Modulo}-15\ \textit{addition}$$

These relations for the last states are implemented by a PLA and a counters. At the beginning of a nodal cycle the I⊕3 PLA implements modulo-15 addition of the content of the I counter and 3 and the LS counter is loaded with this value (LS1), the LS counter is then incremented to generate LS2 and LS3. It was noted by the TPU design team that, if 3 is added to I, adding 11 could be accomplished by inverting the most significant bit of the I⊕3 result.

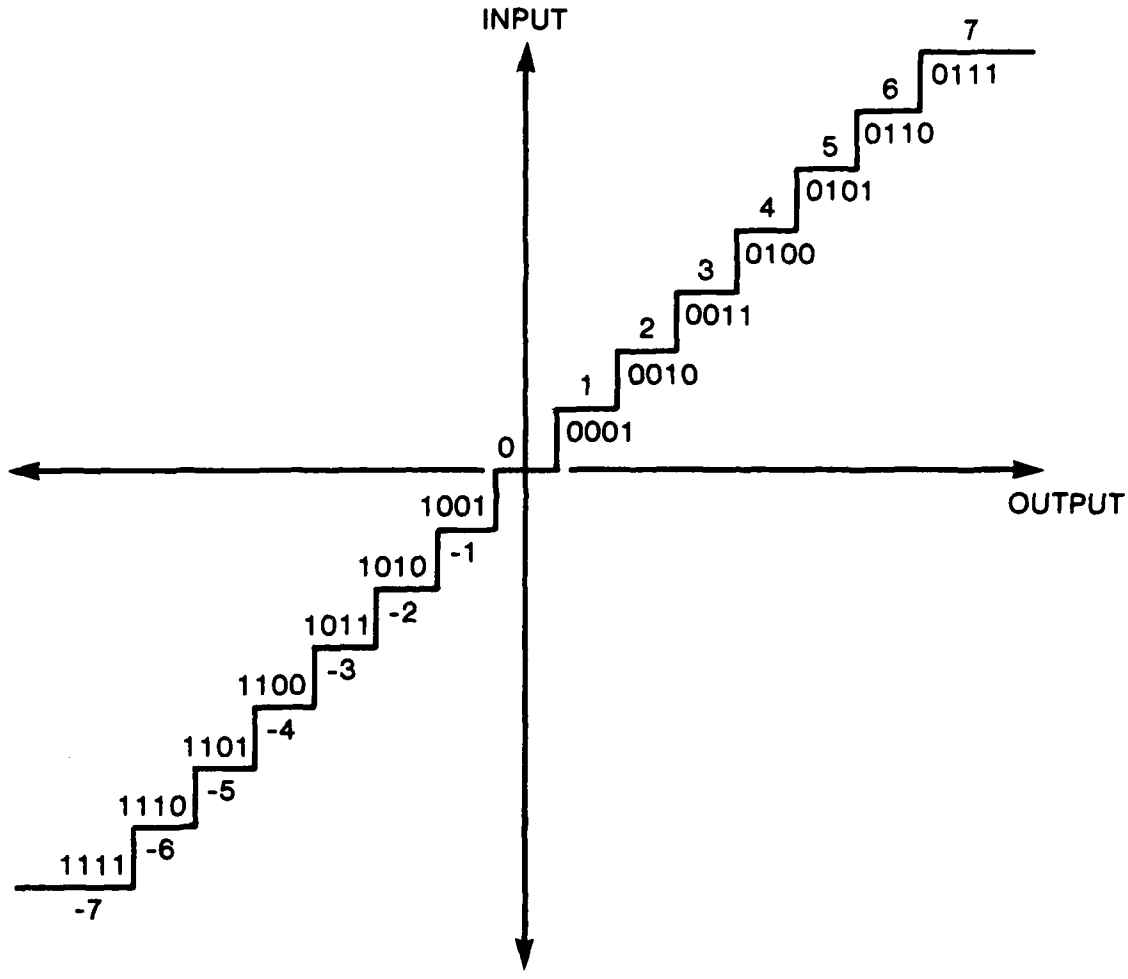


Figure 2.5. Quantizer

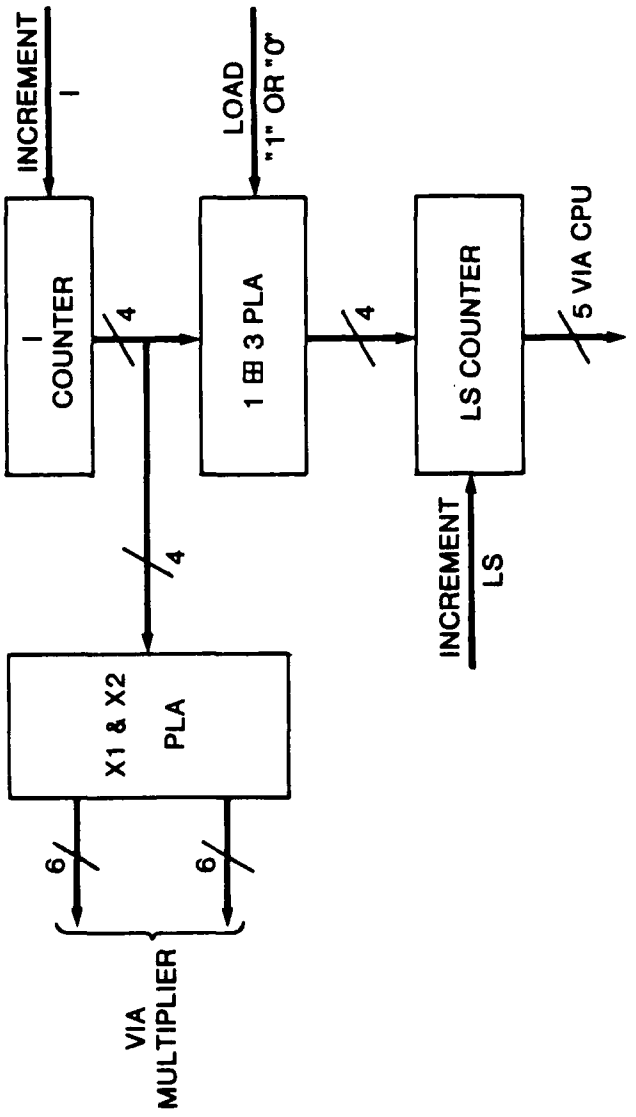


Figure 2.6. TPU

X1 and X2 are generated using a clocked input PLA, where (X1=Cos(LS2), X2=Sin(LS2)) for bm(1), and (X1=Cos(LS5), X2=Sin(LS5)) for bm(0).

2.5 Central Processing Unit

The function of the Central Processing Unit is :

i. finding the "survivor".

ii. normalizing the accumulated metric values.

This unit is the subsystem which performs all the computations required by the Viterbi algorithm. The size of the data path in the CPU is determined by the number of bits presenting accumulated metric value. The magnitude of all the accumulated metric values is always bounded, in our case the maximum spread between the largest and the smallest accumulated metric value is 14 [1]. Thus, the data path size for accumulated metric value is taken conservatively as 6-bits, positive integer in fixed point binary form (radix=2). The accumulated metric values are periodically normalized inside the CPU to avoid any overflow within the data path.

The block diagram for the CPU is shown in Figure 2.7.

2.5.1 Survivor

At every state of the trellis, the "survivor" of the state I is found by adding the branch metric to the appropriate accumulated metric value of the last states, (total of 6 possible), connected to the present State I. The largest among the 6 accumulated metric values is then chosen. This value will be the new accumulated metric value of that state, as shown in Figure 2.8.

The survivor block is composed of a 6 bit register containing the result of the accumulated metric value added to the the branch metric values denoted as Am+bm register, a 6 bit register containing the survivor's accumulated metric value which is set to zero at the beginning of a nodal cycle, a 5 bit register containing the LS (last state) and the D (decoded bit) provided by the TPU corresponding to accumulated metric value read from the memory, a 5 bit register containing the LSS "Survivor's Last State" and Dout "Decoded bit" and the survivor comparator.

The survivor comparator output is a single control line which goes high, if the content of the survivor's accumulated metric value register is less than the Am+bm register, then Am+bm and its corresponding LS and D are shifted in parallel to the survivor's accumulated metric value register and the LSS register. Otherwise, the content of the accumulated metric and LSS registers remains unchanged. Once this process is repeated six times for a present state I, the survivor's accumulated metric value and its corresponding LS and D are sent via the memory. To begin the next nodal cycle, the survivor's accumulated metric value register is cleared. The above procedure is repeated for the next present state of the trellis diagram.

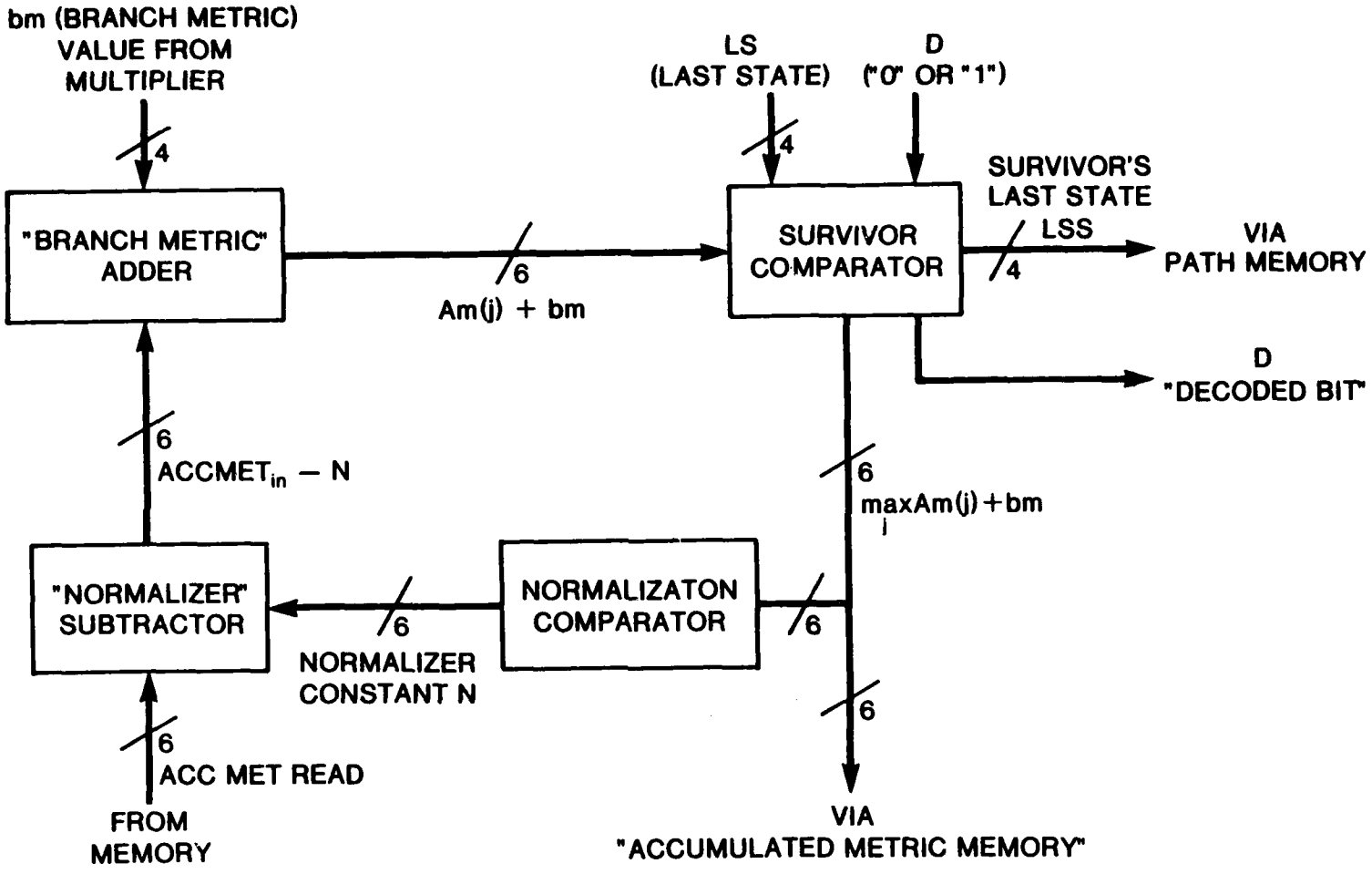


Figure 2.7. CENTRAL PROCESSING UNIT

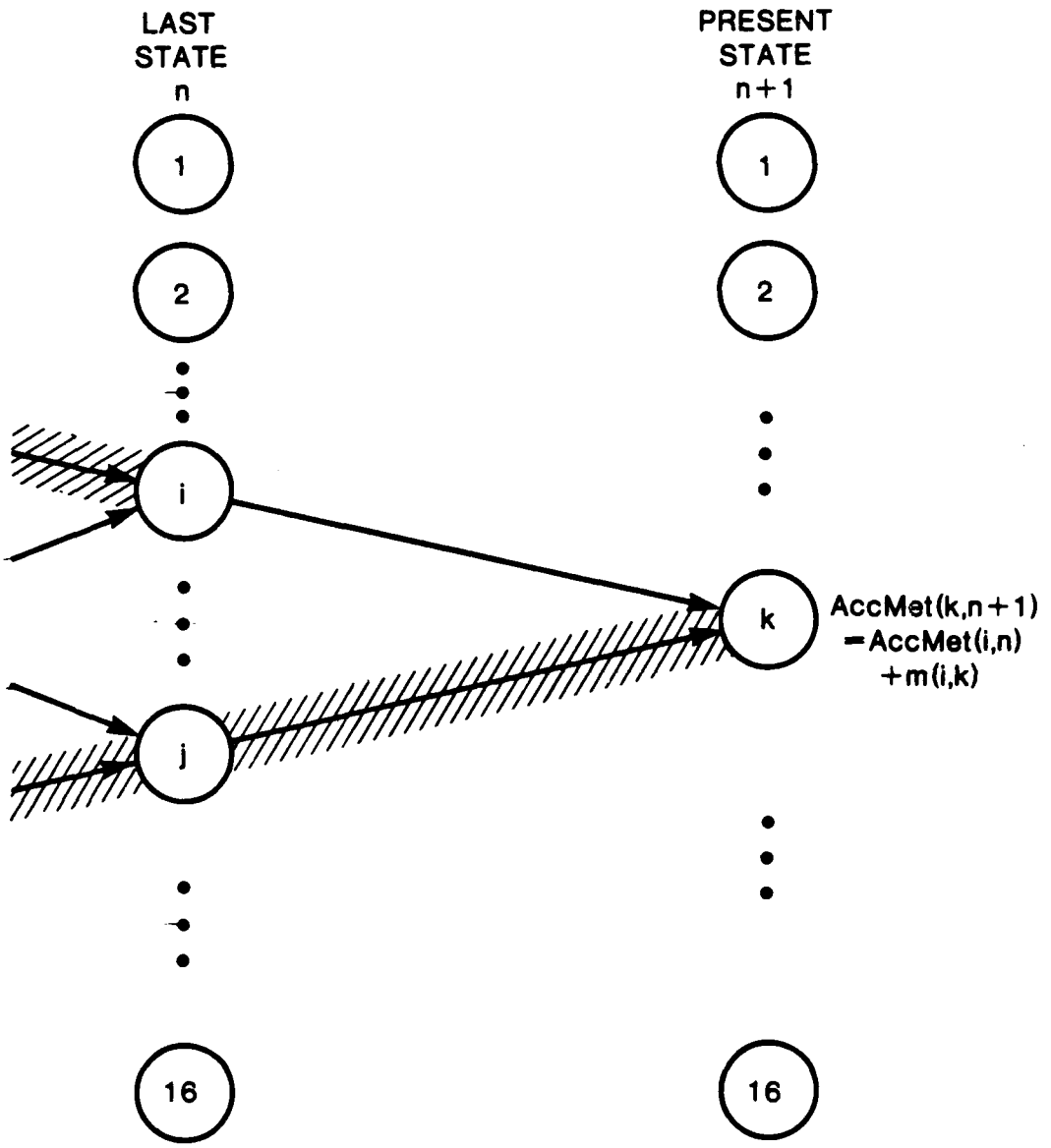


Figure 2.8. Survivor

2.5.2 Normalization

To avoid overflow of the accumulated metric value data path, the normalizer subtracts a constant from all accumulated metric values read from the memory; this constant is found by feeding back the survivor's accumulated metric value to a comparator (normalizer comparator). If the input value to this comparator is less than the normalizer register (initially set to all 1s), the input value shifts in parallel into the normalizer register.

At the end of the symbol cycle, the normalizer register contains the smallest accumumlated metric among all 16 survivor values. Its content is shifted into the register N (initially set to all 0s), and this value is subtracted from all accumulated metric values coming in to the CPU in the next symbol cycle.

2.6 Memory

In order to provide more memory for the data, it was decided not to store the phase values of the paths in the trellis diagram. This can be done without any loss of generality since the Viterbi algorithm takes into account the random phase in the expression for the the branch metric values.

The memory requirement is partitioned into two independent sections: Accumulated Metric Memory and Path Memory.

2.6.1 Accumulated Metric Memory

The main issue in designing this subsystem is the number of 6 bit length registers needed for the Accumulated Metric Memory.

The Viterbi algorithm requires the knowledge of both the accumulated metric value at time nTs and (n-1)Ts. Therefore, a pair of 6 bit registers are used for each state to store accumulated metric value of that state at time nTs (back-up register) and (n+1)Ts (front register). This yields 32 rows of 6 bit registers. The "back-up" registers are addressed by the LS values provided by the TPU; their contents are only READ during each branch cycle. The "front" registers are addressed by the present State I and are used only to WRITE the survivor's accumulated metric value at the present state at the end of each nodal cycle. The contents of the front and are replicated, front to back, at the end of every symbol cycle.

2.6.2 Path Memory

The path memory stores all data bits corresponding to the 16 survivor paths found by the Viterbi algorithm during each symbol cycle. An interesting property of the Viterbi algorithm is the negligible loss of optimality by truncating the paths on the trellis diagram at some fixed lag time NTs and outputting the oldest bit of any of the 16 paths.

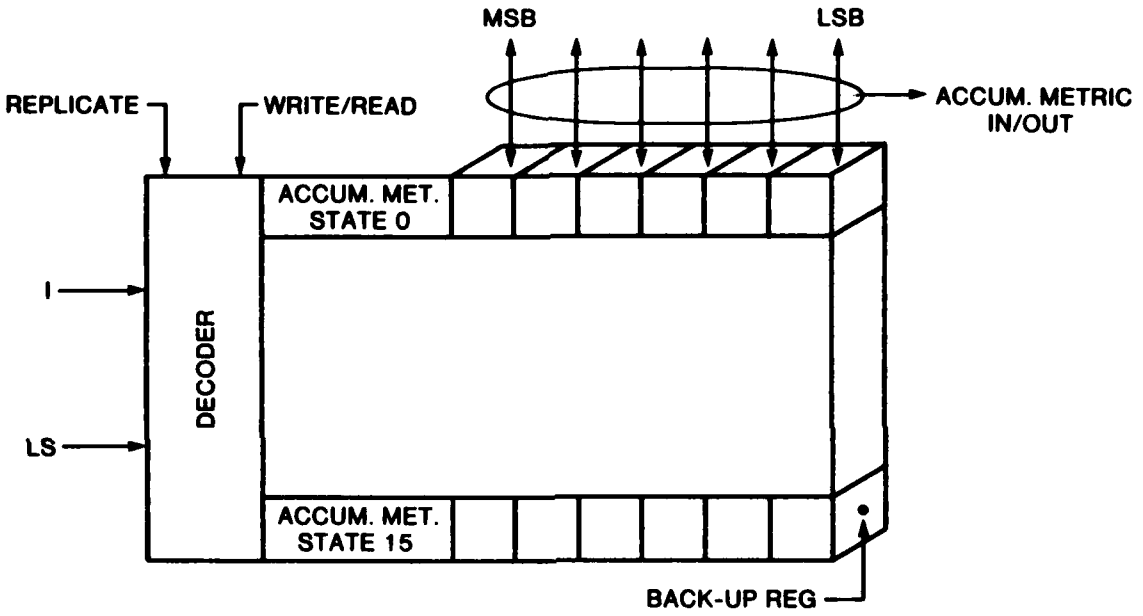


Figure 2.9. Accumulated Metric Memory

2.6.2.1 Path Memory Truncation

The length of each path memory register is dependent on the truncation point N on the trellis, i.e., how many NTs seconds one should have to wait before outputting the oldest bit.

This has been subject of previous research [1], and it is shown that with high probability the best path among all 16 paths could have diverged from the correct path for only a reasonably short span. We have thus taken N=30; for each 16 paths this requires a 30 bits length shift register which stores the decoded bit "D" corresponding to the transition of each survivor.

2.6.2.2 Path Memory Organization

When the flow chart of Figure 2.2.b is sequenced at each state of the trellis diagram, it results in a survivor path which stems from the last state of the survivor connected to the present state I. This is shown in Figure 2.10, in this example the survivor's last state "LSS" for present state 1 and 2 is 4.

The issue here is the number of 30 bit long shift registers needed for the path history. The memory is organized as 32 rows of 30 bit long shift registers as shown in Figure 2.11. These 32 rows are divided into 16 pairs of "back-up" & "front" registers. The front registers are also used as FIFO (First In First Out) registers. The CPU finds the survivor's last state, LSS; the LSS addresses the corresponding back-up path register; its contents are then shifted in parallel to the front shift register addressed by the present state I; and D is then shifted, serially, into the front path register. This is how the path history of the trellis diagram is mapped into the memory. In essence, the back-up is used for what the path "was" at time (n-1)Ts, and the front contains the history of what the path "is" at time nTs.

At the end of a symbol cycle, the content of each front register is shifted in parallel to its corresponding back-up register before starting the next symbol cycle; thus, at the beginning of every symbol cycle, the contents of the front and back-up registers are the same for each state. This defines our *Storage Procedure* refered to in Figure 2.2.b

The special property of this memory is, namely, it's dual shift register with capability of copying in parallel the front register into the back-up register and vice versa. The block diagram of the path memory is shown in Figure 2.11.

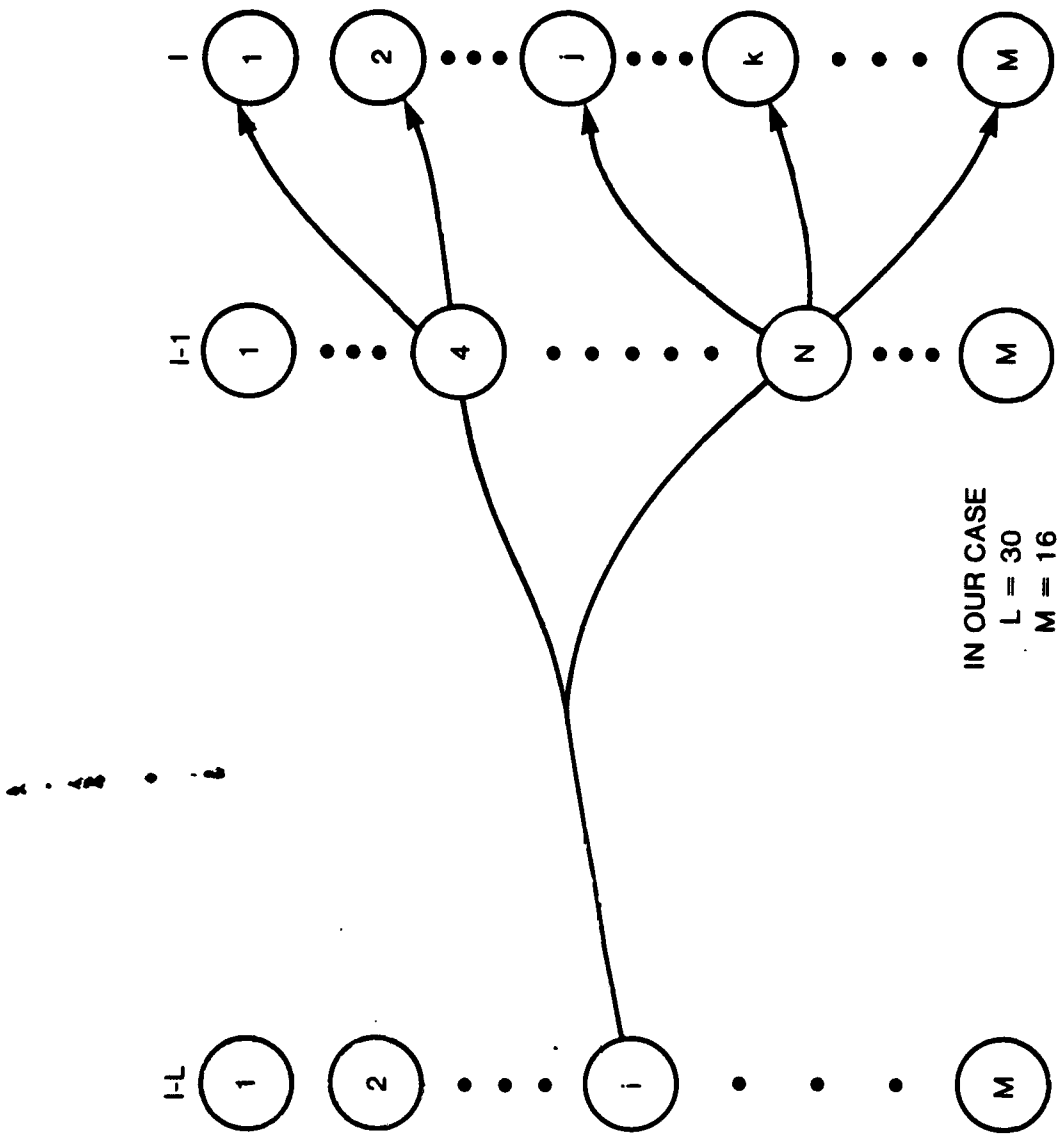


Figure 2.10. Path History

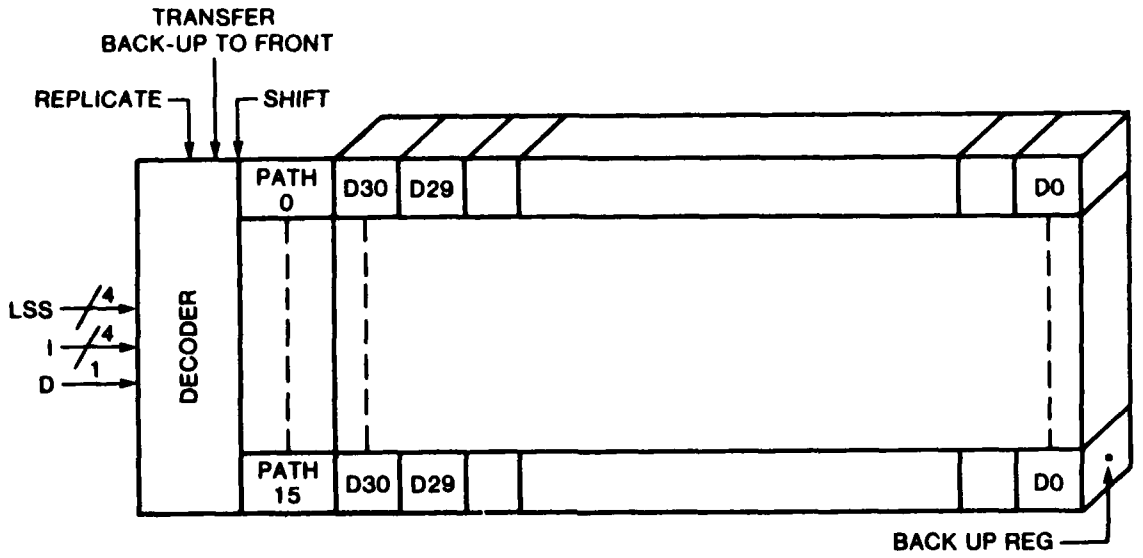


Figure 2.11. Path Memory

2.7 System Architecture

The system architecture is shown in Figure 2.14, which summarizes our system processing and interface requirements.

This chip operates in a pipelined processing mode. One complete cycle of this system is described in this section. It is assumed that everything is RESET to zero, (except normalizer register, which is set at 63).

The inputs to this chip are: $y_c(1)$, $y_s(1)$, $y_c(-1)$ $y_s(-1)$, the In-phase and Quadrature components of bit 1 and 0 quantized to 4 bits each. The TPU provides (X1,X2) pointed by the present state I, so the branch metric generator first computes bm(1) and then bm(0). For every present state I, its corresponding last states and decoded bit D are each outputted via the CPU. The last states are also sent to the the accumulated metric memory. The accumulated metric memory outputs the accumulated metric value from the back-up addressed by its last state via the CPU. This value is normalized and then passed over to the adder; that value is added to the branch metric and passed over to the survivor comparator, which compares this value with zero initially. The above is repeated 6 times. At this point, the survivor comparator register holds the largest value among the accumulated metric, the last state of the survivor, and the decoded bit D, which has caused this transition. The accumulated metric value of the survivor is sent to the accumulated metric memory which is written in the front accumulated metric value register of the present state I. This value is also fed back to the normalizer comparator, which always holds the smallest value it encounters throughout its processing time. The last state of the survivor, LSS, and its decoded bit D are sent via the path memory, which uses LSS as the path address of the corresponding back-up path register. The content of the back-up register is transferred to the front register addressed by the present state, and D is then shifted into that front register (FIFO).

I, the present state is incremented, and the above procedure is repeated 16 times.

At the end of 16 "nodal" cycles, which constitute a "symbol" cycle, the front and the back registers are replicated in pairs, and the normalizer constant is passed over to the subtractor to be used in the next symbol cycle.

2.8 Controller

The CONTROLLER is composed of a stored program in a memory which is sequenced in time. The outputs of the controller are called the control bits. The operations performed within the system are determined by the sequence of control bit patterns supplied by the controller. These control bits are called microcodes and are stored in the control memory.

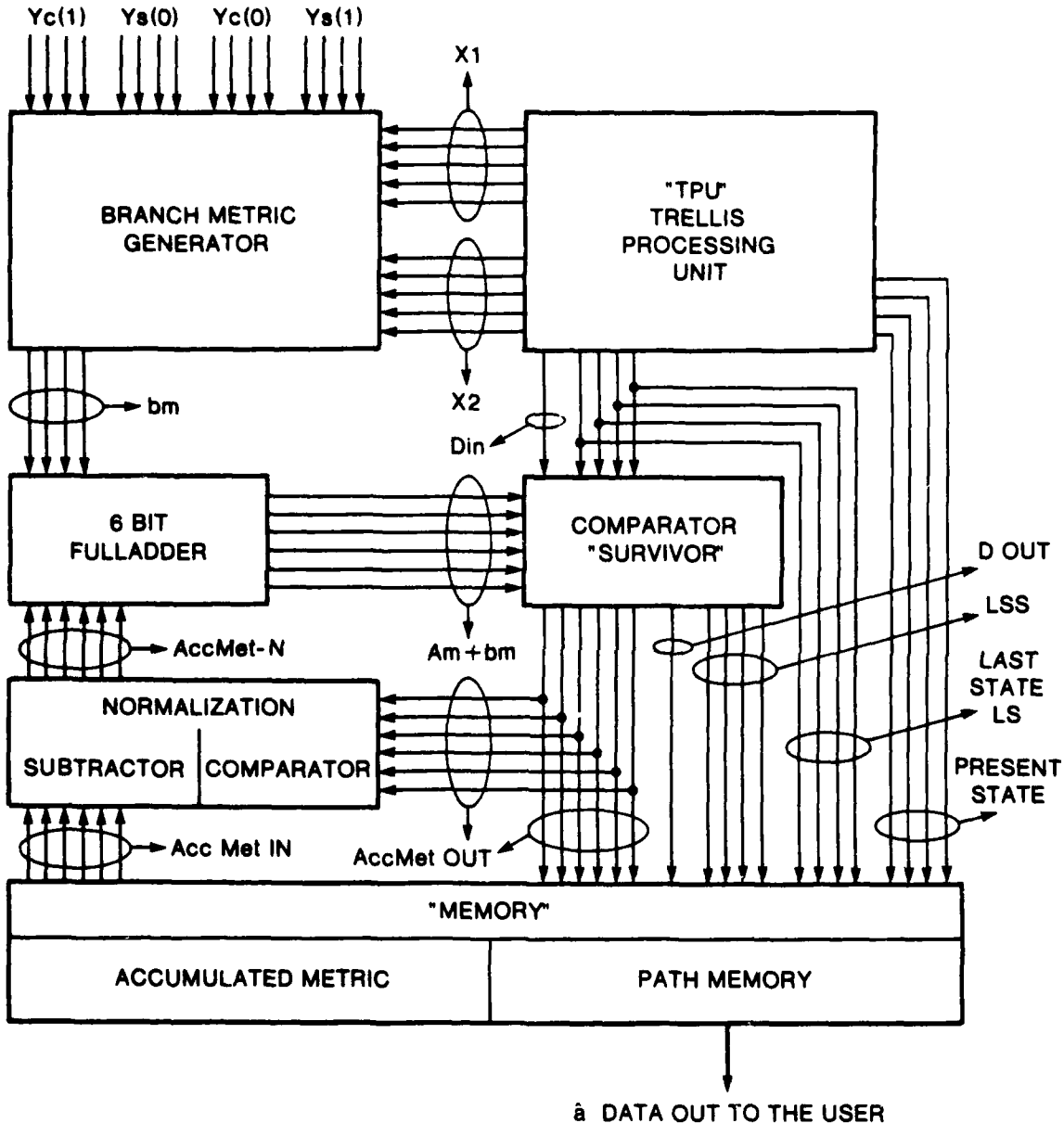


Figure 2.12. System Architecture

Each microcode "word" is defined as the series of stored bits used during each state of the controller[1]. The process is to shift out sequentially each word composed of L bits and use k of these L bits as the state feedbacks to the "micro path," which determines the next state of the controller. The remaining L-k bits are the control bits which will strobe physical points of the system. The pattern of these control bits is determined by the timing requirements of each subsystem. The controller structure was designed at a time when none of these requirements were defined; so, a general design approach was adopted as shown in Figure 2.13.

The instruction PLA contains the microprogram for sequencing all the subsystem. Minimizing the size of the control memory (or the instruction PLA) is equivalent to minimizing the number of words (or states) in the control memory. This objective requires the micro path to handle "Do Loops"[2] to sequence the control memory. To separate the issue of timing and sequencing, the metric-free concept of the sequence domain is used to derive the micro-path for the controller. The flow chart of Figure 2.2.b contains the abstraction for sequencing the Viterbi algorithm.

The condition for a jump at the end of the loop depends on the number of iterations in every loop. The number of iterations in every loop consists of

i. nodal cycle, which contains 6 branch cycle

ii. symbol cycle, which is composed of 16 nodal cycles.

This requires two counters in the micro-path which count the number of iterations for each case. To encode this conditional jump, two bits of every word in the control memory are dedicated to specify the jump using t0, t1. Depending on the state of t0, t1, one of the following happens in the micro path

t0	t1	
0	0	increment present address
0	1	jump to NA at the end of nodal cycle
1	0	jump to NA at the end of branch cycle
1	1	jump to NA at the end of symbol cycle

Here NA stands for the next address field in the microcode word as shown in Figure 2.13.

[1]We shall use direct control scheme, where we provide a dedicated bus for each physical point of the system to be controlled; hence, the length of each word is fixed.

[2] Do Loops in the sense of iterating an algorithm a fixed number of times.

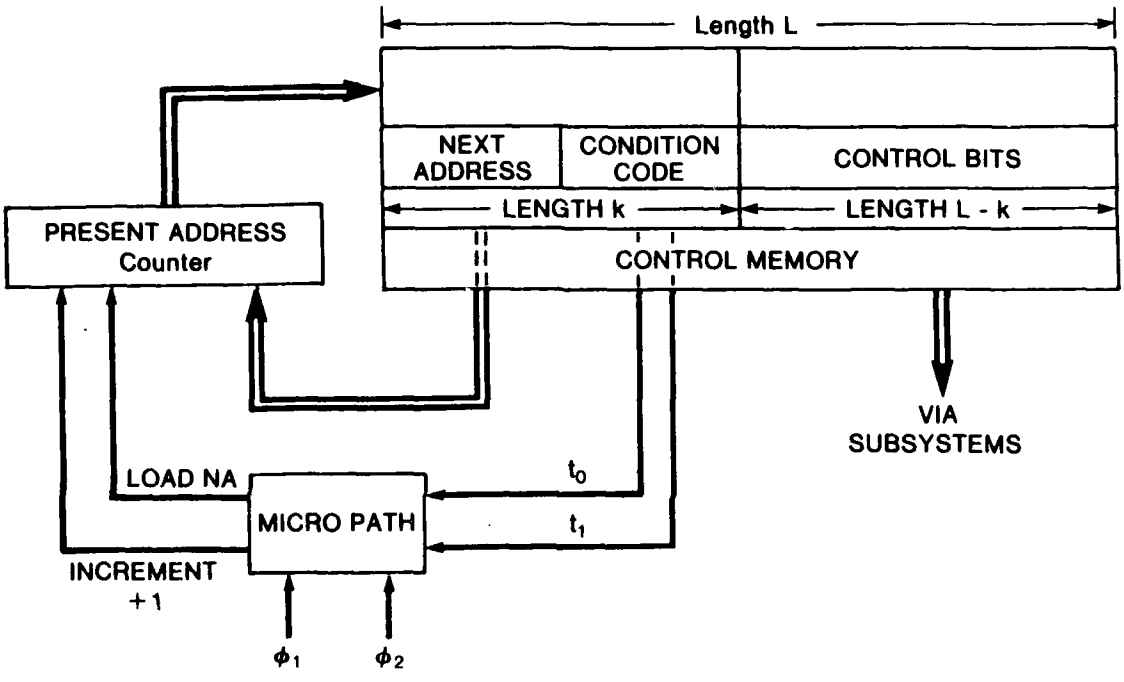


Figure 2.13. Controller

The control bit section of the instruction PLA contains the microcodes included in the Appendix B. The role of each microcode word is to initiate certain functions within the system and will be described in Chapter 7.

2.9 Floor Plan

The floor plan, shown in Figure 2.14, was designed to minimize the length of busses which link different subsystems together. It was aimed to place the higher bandwidth ports as closely as possible. The final design of the floor plan contains NO overcrossing of VDD and GRD lines, enhancing power distribution within the chip.

The floor plan implemented is not optimum, but it was the best choice within the time constraints during the design effort.

2.9.1 Current Requirement for the Subsystems

The electrical current requirements for every subsystem was estimated using both SPICE simulations and hand estimates. All the VDD and GRD busses are designed to handle these current values

Branch Metric Generator	40ma
Central Processing Unit	35ma
Trellis Processing Unit	35ma
Controller	40ma
Accumulated Metric Memory	30ma
Path Memory	100ma

This concludes our discussion on the system requirements of the UCLA Demodulation Engine. We are now ready to investigate the hardware implementation of this system.

The following section is intended solely to cover the variations of this system design and will not be used later.

2.10 Discussion and Variations of the System Design

The critical issues at the system level are addressed in the following sections. Section 2.10.1 focuses on the issues of the architecture of this chip, and section 2.10.2 focuses on the monitoring and variations of the application of this chip as a part of a digital communication receiver.

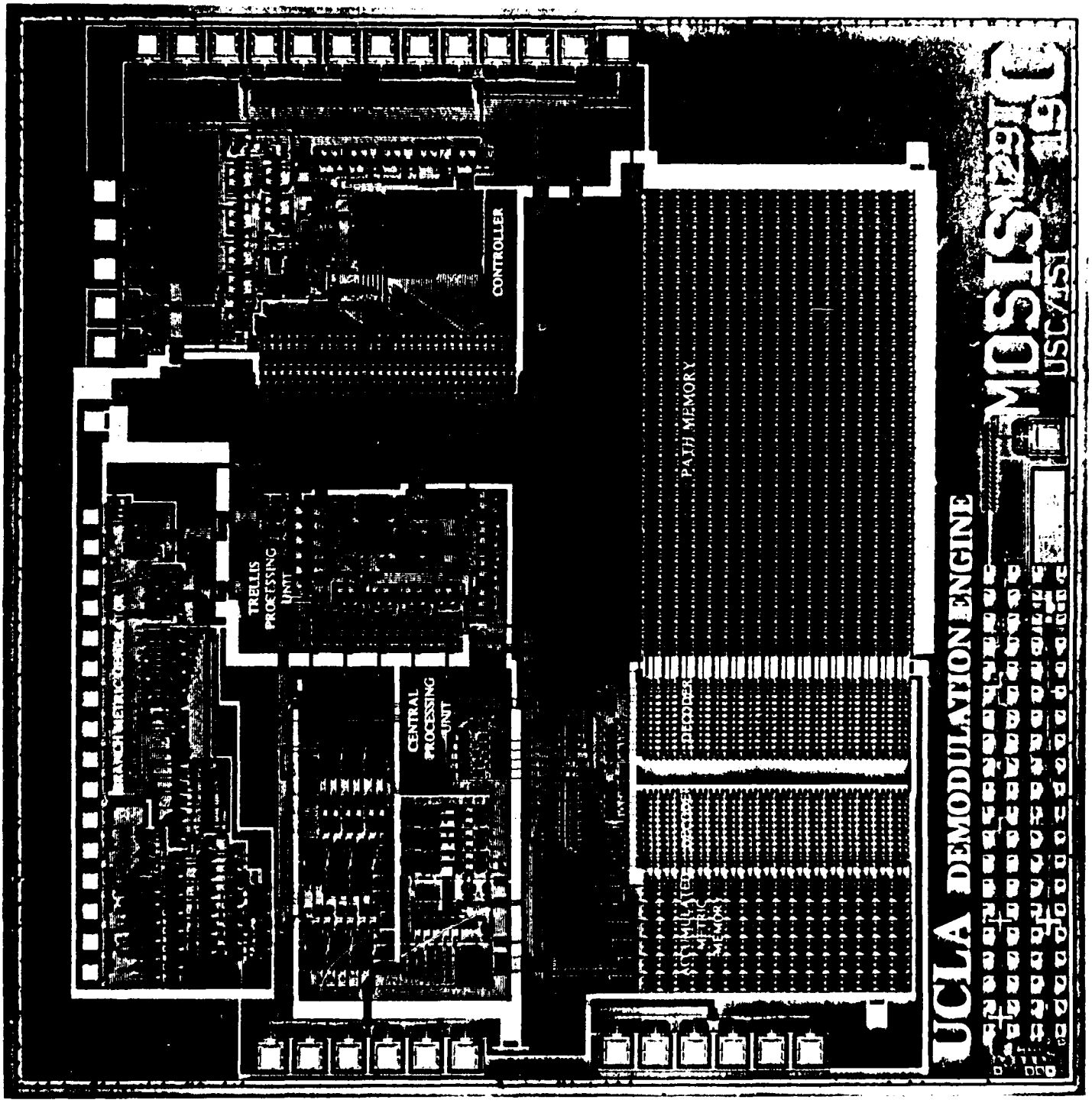


Figure 2.14. Floor Plan

2.10.1 Architectural Trade-Offs

I. Serial Processing

The main goal regarding architecture for this chip was to fit everything on a single die. This objective was met only at the cost of serial processing.

II. Parallel Processing

To increase the overall speed of the Viterbi algorithm, parallel processing becomes necessary. In a full parallel Viterbi algorithm processor, all metric calculations are processed in parallel, and accumulated metric values and the path memory are updated at once. It can be roughly estimated that a full parallel processed 128-state Viterbi decoder would require an order of 200,000 transistors, which is beyond the capability of today's VLSI technology. A compromise between serial and parallel architectures must be made to transform the system into a single chip.

There are some properties of the trellis diagram which are advantageous to a parallel architecture. The so-called "butterfly" operation can be used to simultaneously compute the branch metric values of multiple numbers of states on the trellis diagram.

III. Bit Slice Architecture

A bit slice design is a highly structured architecture which can be expanded into an n bit processor with minimal waste of area and can be easily tested.

When soft decision decoding is used, it normally requires the branch metric generator block to compute the branch metric values. This block occupies a reasonable area of the total available space; therefore, it cannot be repeatedly used for every slice.

When hard decision decoding[1] is used, the bit slice design can be quite useful since, in this case, the branch metric calculations are simple to perform.

The challenging design problem in the bit slice architecture for the Viterbi decoder is the interconnection network for the slices; it may be possible to simulate the trellis diagram's structure by using direct links among the slices. This approach eliminates the need for a separate block (TPU) providing the necessary information embedded by the trellis diagram.

IV. Timing and Controller

[1] In hard decision, observed vectors are binary bits, and the branch metric values are Hamming distances.

A different approach to timing and the architecture of the controller was to design self-timed [2] blocks at the subsystem level. The complex problems of topological organization and interface of the subsystems for a self-timed system design were the main reasons this approach was not considered.

V. Fault Tolerance

In real time applications, uninterrupted operation is essential, and the chip must perform reliably. Error detection and correction logic can be used effectively to decrease the probability of failure. However, fault tolerance was not considered in designing this chip because the physical space needed by this type of circuitry seemed prohibitive.

2.10.2 System Application of the Chip

On board a modern digital communication system, it is natural to assume availability of a microprocessor (e.g., Z80,MC6800) for an inexpensive mobile radio receiver or a medium size processor (e.g., LSI 11/23) for a communication satellite system. This processor can be used to supervise the demodulation process performed by UCLADE. The capability of executing sophisticated testing algorithms to take full advantage of the outputs provided by this chip depends upon the processing abilities of the supervisor. Output information can be used to both monitor and test the chip. These inputs and outputs as shown in Figure 2.15 are

1. Reset-1 bit resets all subsystems.

2. Freeze-1 bit freezes the controller state; all activities in other subsystems are halted.

3. Test Clock-4 bits can be used for level-sensitive scan design techniques available in TPU and controller. (Shift signal, inputs and outputs are used to shift in test values in the program counter in the controller).

4. Control Signals (CTS)-8 bits, 5 bits are the address lines to the set of microcodes executed by the controller.

5. Present State (I)-4 bits is the state for which the chip is presently finding the survivor.

6. Survivor's Last State (LSS)-4 bits is the survivor's last state for the present state I.

7. Accumulated Metric Value-6 bits is accumulated metric value of the survivor.

8. Normalizer Constant-6 bits can be used to monitor the ... ge of accumulated metric values.

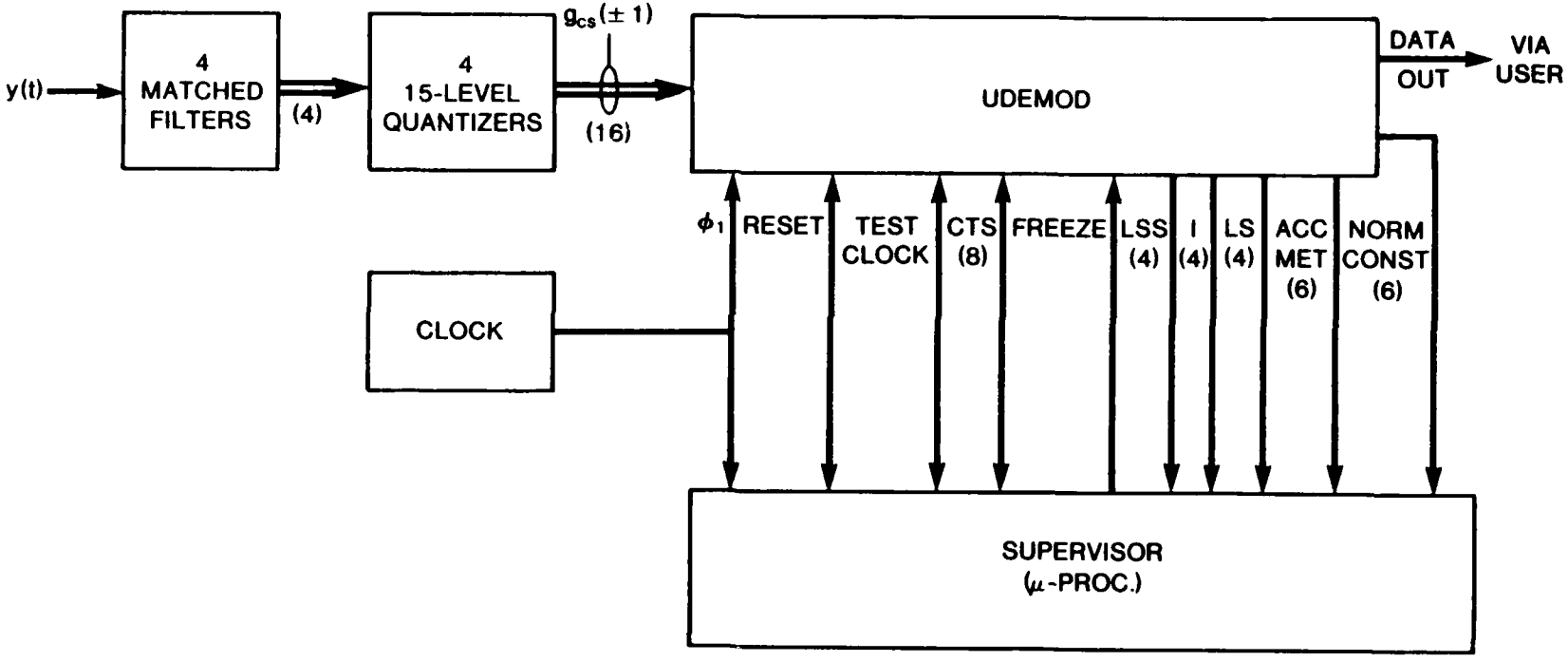


Figure 2.15. Supervising UCLADEMOD

Preferably, multiple numbers of this chip are used and operated concurrently; hence, the supervisor can, by monitoring the outputs from a set of UCLADE chips, use polling algorithms to switch to the "operating" chip.

A discussion of real time testing of the chip using the test clock for level sensitive scan testing is included in chapter 7.

2.10.2.1 Closed-loop Carrier Phase Tracking Application

Recently, researchers [13,14] have suggested using the Viterbi algorithm processor in a closed loop system for tracking and acquisition as opposed to our usage of the open loop application, which assumes the carrier is fully synchronized. The phase values estimated by the Viterbi algorithm can be used by a decision-directed phase lock loop to adjust the correlators phase values and carrier frequency.

As stated earlier in sec.2.6, it was decided not to store decoded phase values, but as a result of our approach to simultaneous phase and data estimation, the TPU can be modified to output phase values via the path memory.

2.10.2.1 Modification for Other Applications

The Viterbi algorithm is used for convolutional codes, inter-symbol interference channels and so on. The differences among these applications are only in the computation of the branch metric values and the connectivity of the trellis diagram. Hence, the basic architecture of this chip remains intact, but the branch metric generator and the TPU have to be modified for the particular application.

A Note on This Report

1. Simulation, at the subsystem level was not possible because of lack of appropriate simulation software at UCLA. The only simulation results available at this time are transient time simulation results for path delay calculations by SPICE, included in this report.

CHAPTER 3
Branch Metric Generator

Contributors:

Tim Broadnax
Erich Huang
Judith Chou

3.1 Project Description

The branch metric generator generates branch metric values as its outputs. The operation required to compute bm(1), bm(0) is a two dimensional vector inner-product using the expression

$$bm(1) = Yc(1)X1 + Ys(1)X2$$
$$bm(0) = Yc(0)X1 + Ys(0)X2$$

Inputs to the branch metric generator from the TPU are X1, X2 and Yc(1), Ys(1), Yc(0), Ys(0) are system inputs, denoted as $(y_{nc}(1), y_{nc}(0), , y_{ns}(1), y_{ns}(0))$ in Chapter2. X1 and X2 are each five bits (1 sign, 4 fraction); Yc and Ys are each 4 bits (1 sign, 3 magnitude). Each pair of Yc and Ys is multiplexed, one at a time; therefore, the multiplier computes bm(1) and then bm(0) (4 bits magnitude).

3.2 Implementation

The basic tradeoff in designing the multiplier was speed vs area. The design group felt that the fastest possible implementation would be a Booth multiplier; however, the space necessary seemed prohibitive. A good compromise was found by using the Carry Save Adder Scheme (CSA).

For positive inputs, the implementation is standard shift and add. The following example illustrates the multiplication operation:

			multiplicand	
multiplier	00011	(3)	00011	1
multiplicand	101	(5)	00000	0 +
product	001111	(15)	01100	1
			01111	

The first bit of the multiplicand is tested; if it is a 1, the multiplier is added to the partial product, originally 0. If the second bit is 1, the multiplier is shifted left (multiplied by 2) and added to the partial product. The process is repeated for all 4 bits.

The CSA tree is shown in Figure 3.1. The branch metric generator performs the vector multiplication in parallel, and products are combined. This results in only one level of Carry Lookahead Adders; thus, faster operation can be expected.

For non-positive inputs, the two sign bits are Xored together. An output of 1 indicates that the 2 inputs are of different signs, and the product will be negative; in this case, 2's complement arithmetic is employed *. If the multiplicand bit is 1, each multiplier bit is inverted before being added to the partial product, and an appropriate carry is added since negative numbers are realized by inverting all bits and adding 1.

The following example illustrates this procedure.

			multiplicand	
multiplier	100011	(-3)	1111100	1
multiplicand	101	(5)	0000000	0+
product	11110001	(-15)	1110000	1
			101	
			1110001	

The resulting logic diagram is shown in Figure 3.2.

In one row of CSAs, 7.5 is added to the output. The factor of 7 was discussed in Chapter 2, and the .5 rounds off the truncated output. The logic diagram for the Branch Metric Generator is shown in Figure 3.3.

It should be pointed out that, due to the functions easily realized in MOS, the gates in this diagram are NOT exactly the ones on the chip, but the logic is unchanged. For example, the XOR-AND combinations are realized with an XNOR-NOR combination since this implementation is faster and requires less space.

*It doesn't matter what arithmetic is used since the final output bm is always positive.

3.3 Cells

1. Input pads & Multiplexers: The input multiplexing, (Ys, Yc), is achieved by the input pads. The pads are 100 λ wide and 106 λ long. The multiplexers are directly beneath the pads as shown in Figure 3.5.

2. Input Gating: The gate cells implement the circuit shown in Figure 3.6. In order to generate W and V signals for latter use, it takes X bits from the bottom and the sign of Y bits from the right side. The W signals will be inputs to the AND gates, which feed the adders. The V signal gets ANDed with the multiplier for carries and sign extension.

3. AND Gates: The AND function is implemented with a NOR gate which takes inverted inputs.

4. Full Adder: The most important cell of this block is the full adder. This cell is used 33 times. It has 3 inputs and 2 outputs (sum, carry). The transistor-level implementation used is found in Carr & Mize's book [11] as shown in Figure 3.7.

5. Half Adder: The half adder uses the same basic scheme as the full adder. It has 2 inputs and one output (sum) as shown in Figure 3.8.

6. Carry Lookahead: The multiplier group decided to use carry lookahead instead of Manchester carries because it was felt that, by doing so, less space would be used. Because it is reasonably fast without being overwhelmingly complex, the lookahead is generated 2 bits at a time.

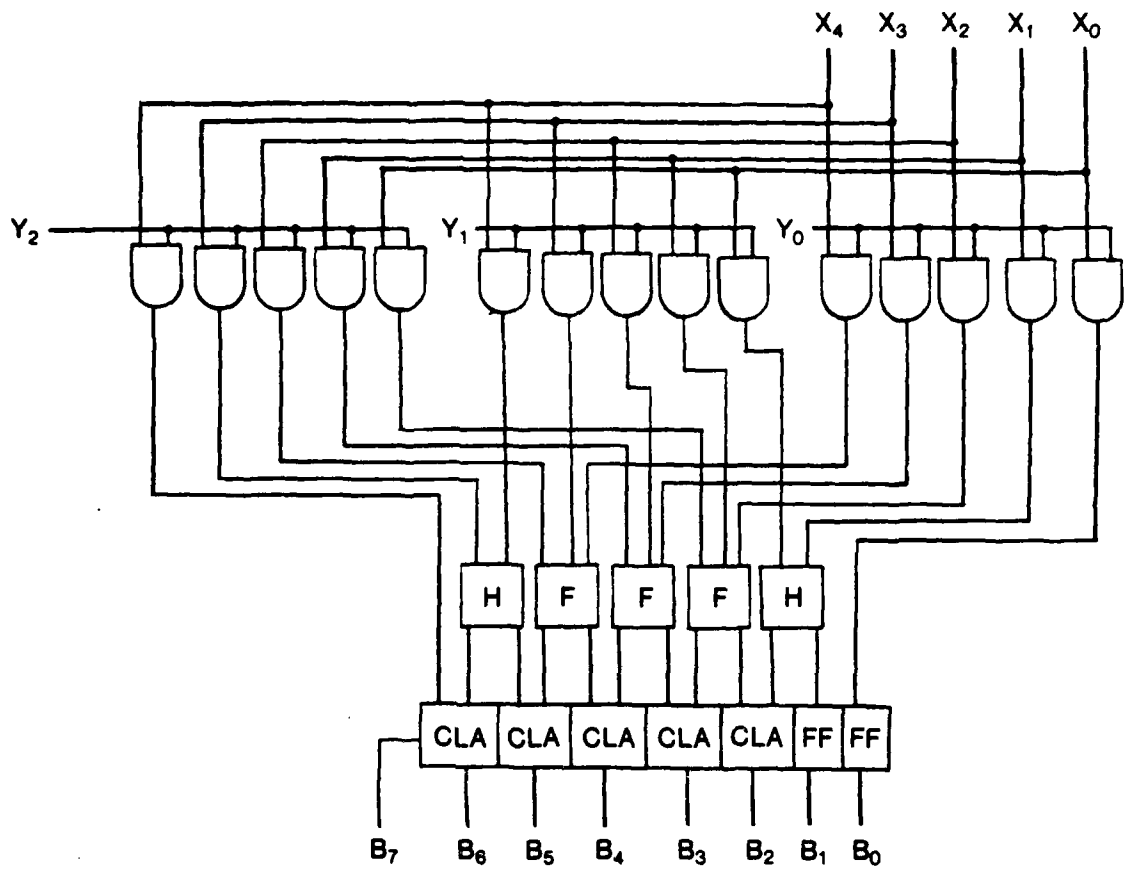
   There are actually 3 different lookaheads cells. The first takes 4 inputs, 2 for each bit position, and generates Cout. The next cell takes this signal and 4 additional inputs and generates Cout. The last cell takes these 2 inputs and produces a carry for the final add.

7. Overall chip: The floor plan, shown in Figure 3.9, shows the overall dimensions. The top width is bounded by the number of input pads, 16 resulting in a width of 1600 λ. After the initial input logic, however, the multiplier narrows to 500 λ.

3.4 Timing Analysis

Based on several SPICE runs made on the simple building blocks of this subsystem, it was deduced that it would take 25 nsec for an input to ripple through input gating. It will take 20 nsec for signal to pass through a full adder, and there are 5 layers of full adders.

The final estimated delay for this subsystem is 170 nsec.



H - HALF ADDER
F - FULL ADDER
FF - FLIP FLOP
CLA - CARRY LOOKAHEAD ADDER
B=XY

Figure 3.1. CSA Tree of a Positive Multiply

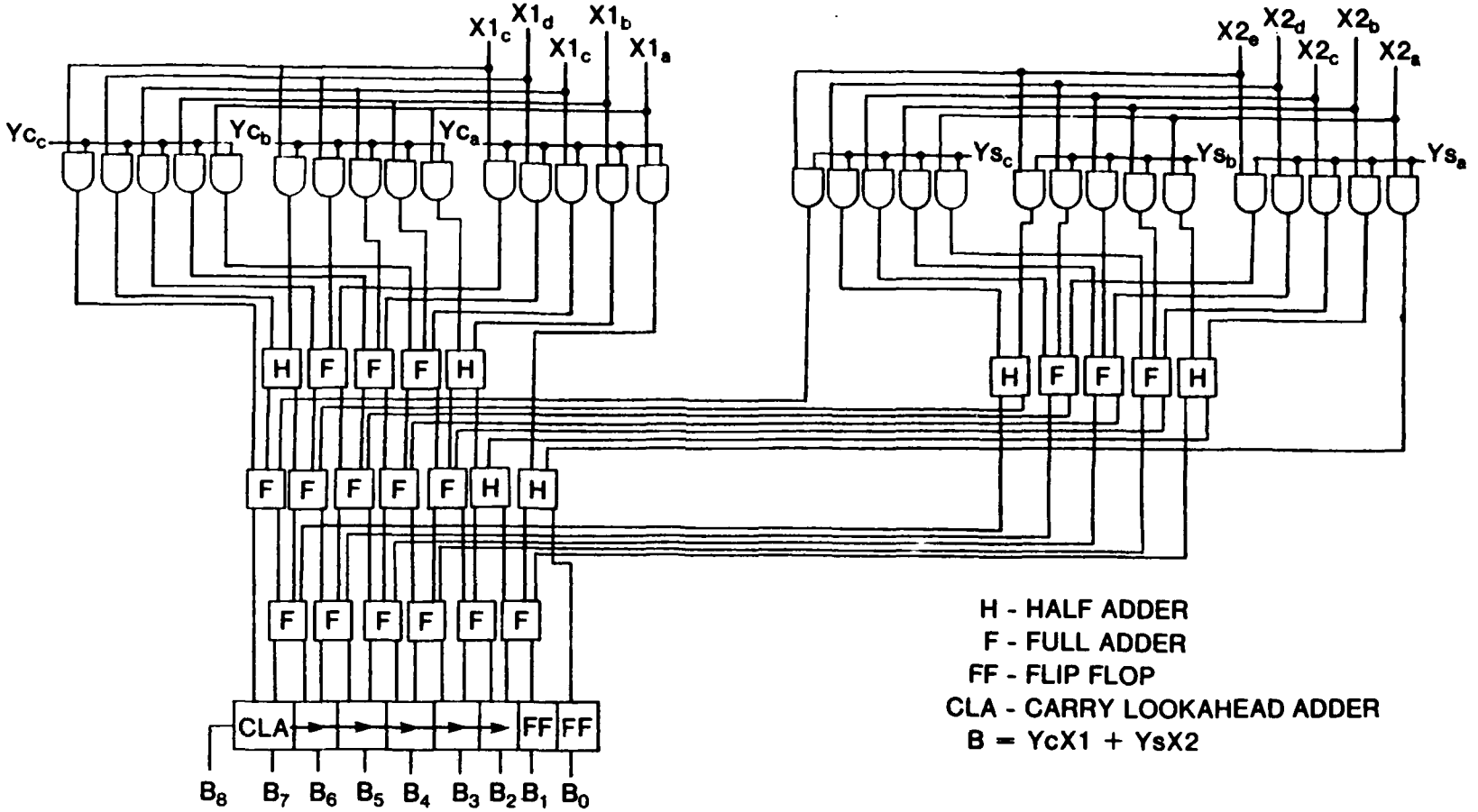


Figure 3.2. CSA Tree of a Positive Vector Multiply (Plate B)

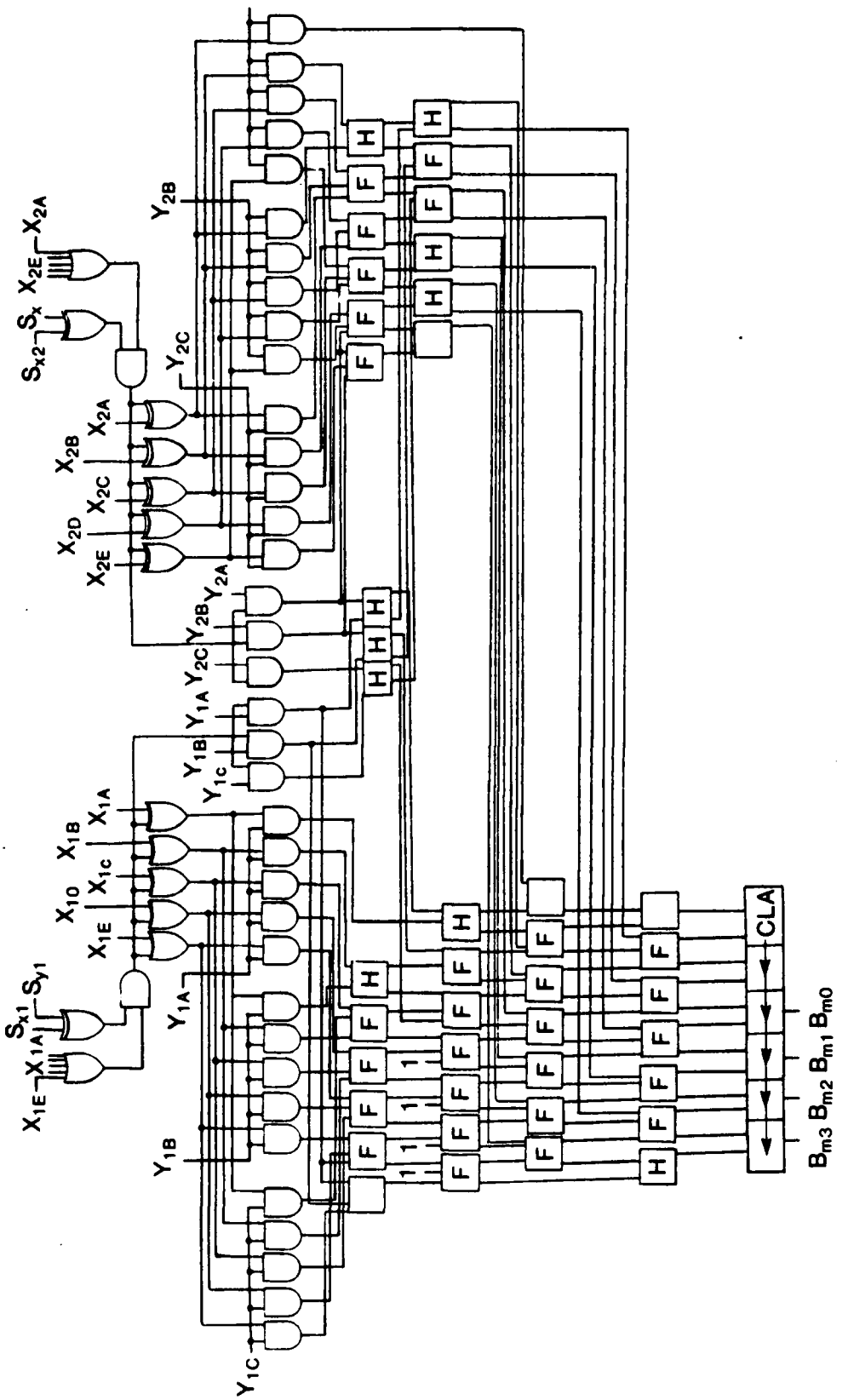
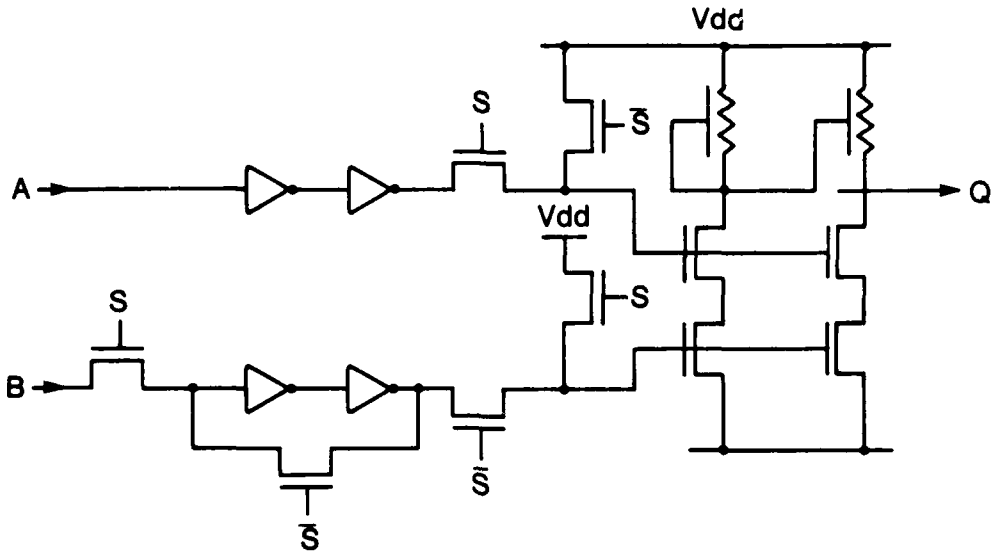
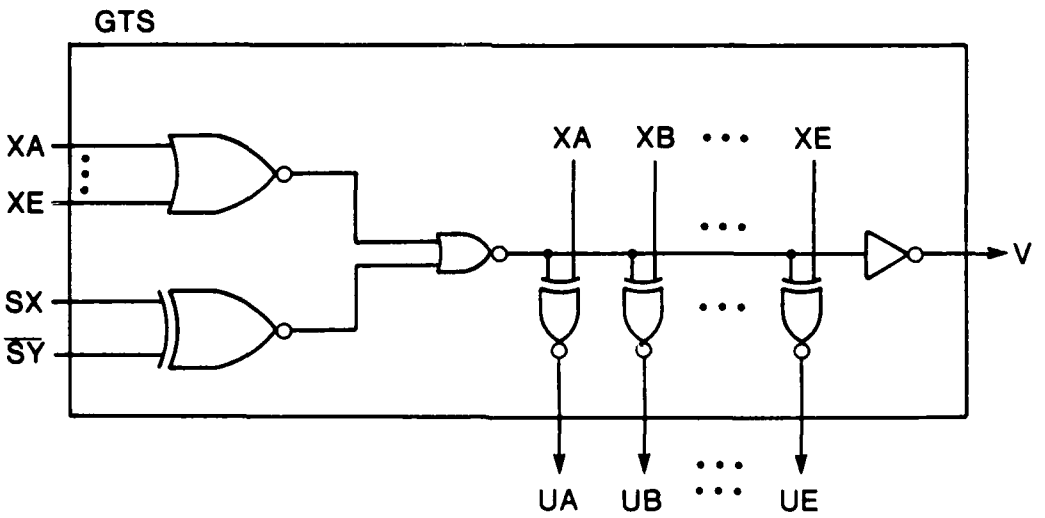


Figure 3.3. Logic Diagram of the Branch Metric Generator



$$Q = \overline{SA + \overline{S}B}$$

Figure 3.4. Multiplexor



$$V = \overline{[XA + \ldots + XE]} + \overline{(SX \oplus SY)}$$

$$U_i = \overline{V} \oplus Xi$$

Figure 3.5. Input Gates

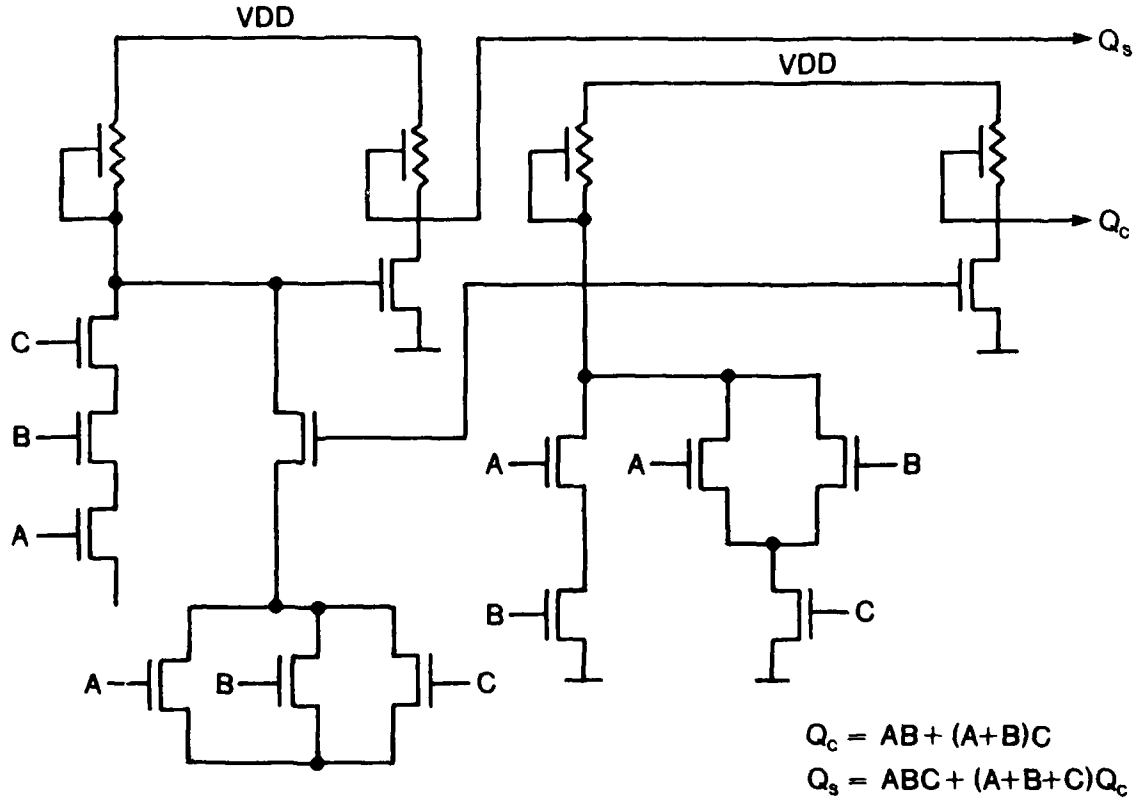


Figure 3.6. Full Adder

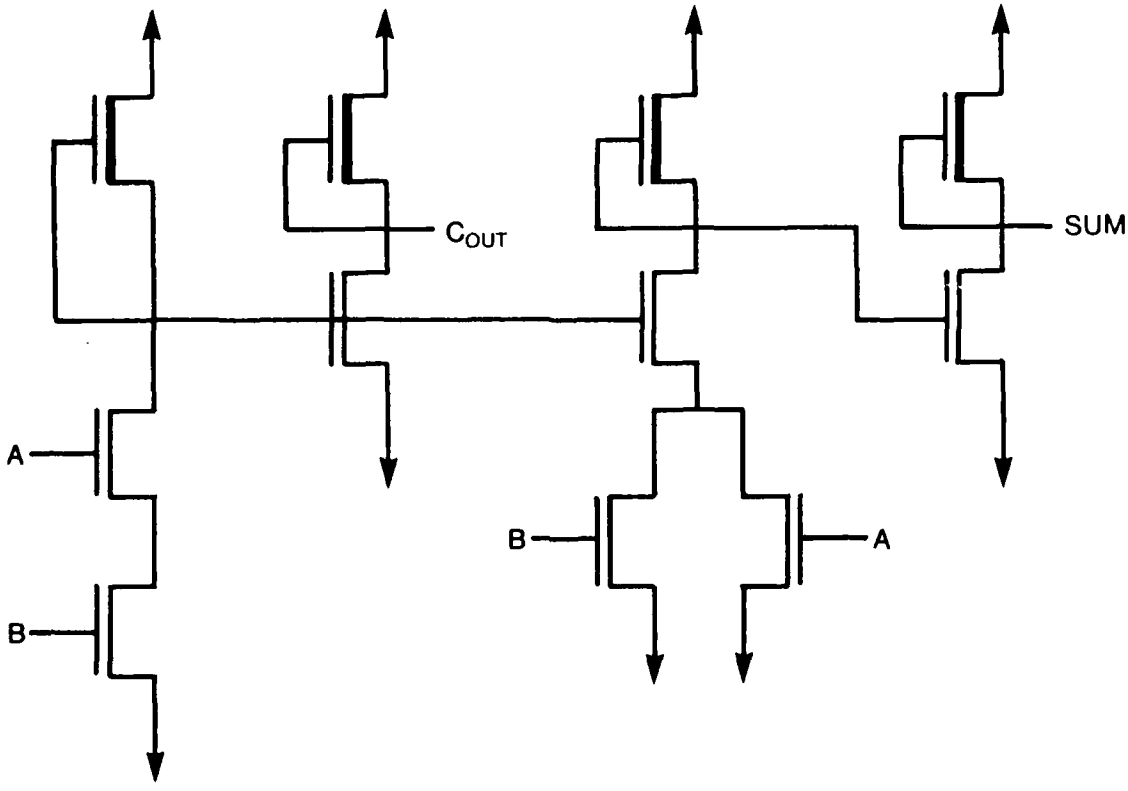


Figure 3.7. Half Adder

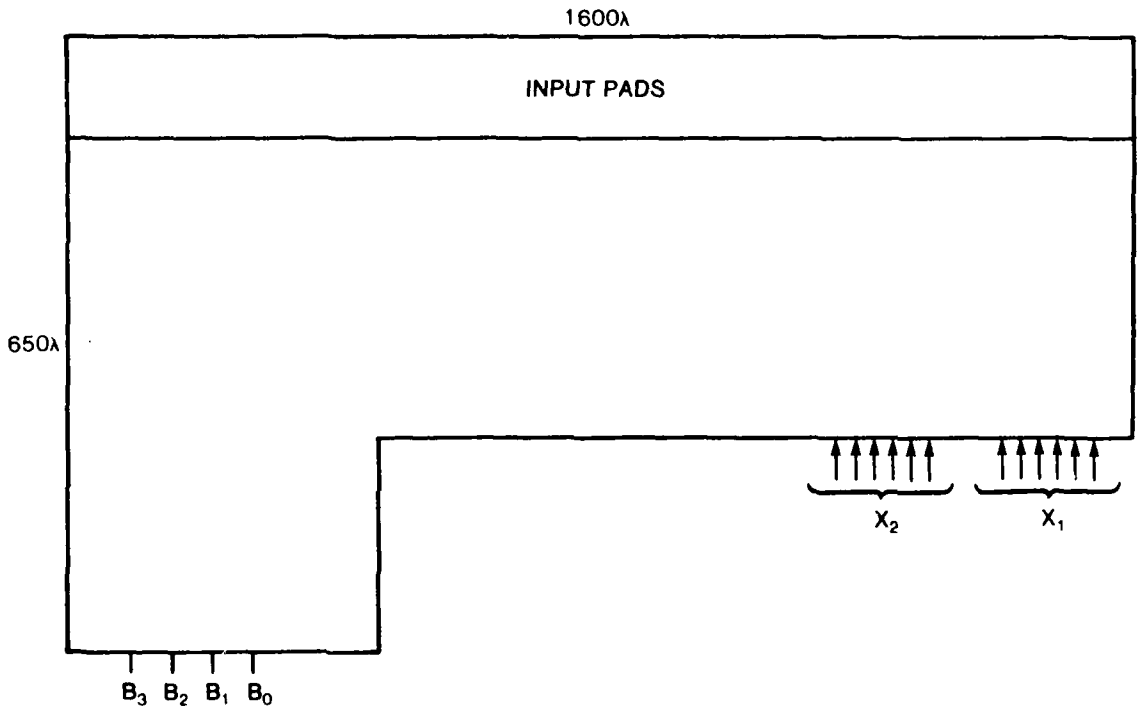


Figure 3.8. Floor Plan (Plate J)

CHAPTER 4
Central Processing Unit

Contributors:

Farshad Meshkinpour
Kameyar Varzandeh
Larry Fitzsimmons

4.1 Project Description

The block diagram for the CPU is shown in Figure 4.1. Detailed operation of the CPU was discussed in chapter 2. In the CPU, all the numbers in the data path are represented by 6 bits positive numbers, and all operations are performed using 2's complement number system. (2's complement is selected so that subtraction can be implemented by using simple adder.)

To summarize the functional requirement of the CPU, an adder is needed for adding accumulated metric value to the branch metric values bm (1) and bm (0); a subtractor is required to subtract the normalizing value, and two comparators are used -- one for finding the constant for normalization and the other to find the survivor.

The floor plan of the CPU is shown in Figure 4.2.

4.2 Implementation

Two options for the architecture of this sub-system are available:

i. to use a central arithmetic logic unit (ALU) which performs addition, subtraction and comparison. This approach would require various registers and a complex data path so that one operation is performed each time and the result routed to the proper register.

ii. to use dedicated cells so that various operations can be done simultaneously within the data path. This approach would require a multiple number of adders, subtractors and comparators.

The second option was selected by the CPU group since building an adder and converting it to a subtractor and comparator provides an easier, faster and more structured design, although the first option would occupy less area.

The system RESET in the CPU interrupts all operations and clears all registers except the Normalizer register, which is reset to all one.

4.2 Cells

The hierarchy of the cells in the CPU is organized using 5 basic cells. The parent cells are listed below, and their siblings are then described. The layout of each parent cell is shown in Figure 4.3.

1. 6 bit Adder -- This cell consists of the following:
   a. input adder cells;
   b. carry generation and carry cells;
   c. output cells.
2. Subtractor -- This cell consists of the following:
   a. complete adder circuit, except that carry in to the lsb is a logic 1;
   b. inverter cells to complement the B input.
3. Comparator -- This cell consists of the following:
   a. input adder cells;
   b. inverter cells to complement the B input;
   c. modified carry generations and carry in cells.*
4. carry generation -- The CPU utilizes 2-bit slice design for carry generation. Internal lookahead was accomplished for the initial carry out. Carries ripple from slice to slice. General carry lookahead Boolean equations for a 4 bit slice are

$$C_0 = C_{in}$$
$$C_1 = G_0+P_0C_0$$
$$C_2 = G_1+P_1G_0+P_1P_0G_0$$
$$C_3 = G_2+P_2G_1+P_2P_1G_0+P_2P_1P_0G_0$$

* The carry generation cell is modified by deleting 1 inverter, 1 nor gate and 8 polysilicon outputs.

$$C_4 = G_3+P_3G_2+P_3P_2G_1+P_3P_2P_1G_0+P_3P_2P_1P_0G_0$$
$$C_5 = C_{out} = G_{out}+P_{out}C_0 = C_{in} \quad \textit{into the next slice}$$

where: $P_0 = A_0 \oplus B_0 or A_0 \oplus B_0 \qquad G_0 = A_0B_0$
$P_1 = A_1 \oplus B_1 or A_1 \oplus B_1 \qquad G_1 = A_1B_1$
$P_2 = A_2 \oplus B_2 or A_2 \oplus B_2 \qquad G_2 = A_2B_2$
$P_3 = A_3 \oplus B_3 or A_3 \oplus B_3 \qquad G_3 = A_3B_3$

Sum output equations are:

$$S_0 = A_0 \oplus B_0 \oplus C_0 = P_0 \oplus C_0$$
$$S_1 = A \oplus B \oplus C = P_1 \oplus C_1$$
$$S_2 = A \oplus B \oplus C = P_2 \oplus C_2$$
$$S_3 = A \oplus B \oplus C = P_3 \oplus C_3$$

The logic diag for this cell is shown in Figure 4.5.
The selected carry generation scheme uses the 2-bit slice design as shown in Figure 4.4. This approach was selected because it minimizes the area and utilizes simple pass transistor logic to generate P & C in parallel. This cell is 76 λ long and 52 λ wide.

5. Inverters -- The A-B inverter is designed for generating both polarities of A and B input signals. It is attached to an adder when the adder is used as subtractor or comparator. It is also used in the P and G generation. It is 35 λ by 36 λ.

6. P, G and S generation -- To implement the propagate and generate signal of the CLA, the circuits are shown in Figure 4.4. A basic XOR gate is used to implement these functions.

4.3 Timing Analysis

To estimate the operating speed of the 6-bit carry generation cell, which is composed of 2-bit slices, were made for the A+B, A-B and comparison circuit designs. These estimates represent the worst case delays using SPICE and hand estimates as shown in Figure 4.6.

I. A+B Mode

The total delay for propagating a logic "0" from slice to slice is 68t, using t = 0.3 nsec; the speed is in order of 20.4 ns for addition.

II. A-B Mode

The total delay for propagating a logic "0" from slice to slice is 73t, using t = .03 nsec; the speed is 21.9 nsec.

III. Comparison (6-Bits design)

In this mode, loading is reduced because many outputs used in addition are not used. The carry from the last slice is used as a flag. The total delay is 39t; therefore, the speed is 11.7 nsec.

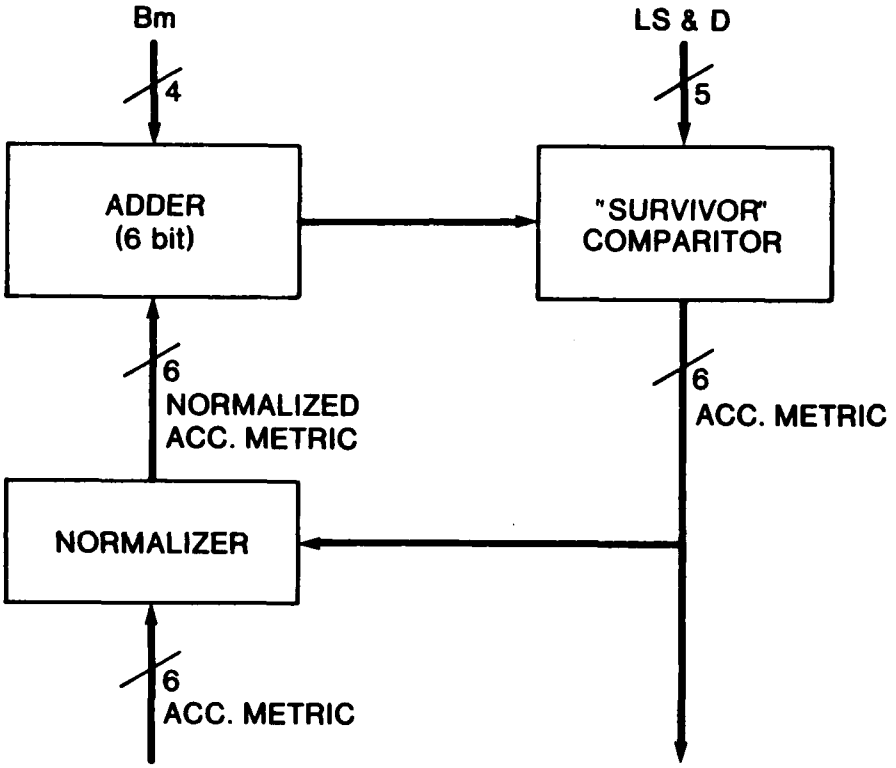


Figure 4.1. CPU Block Diagram

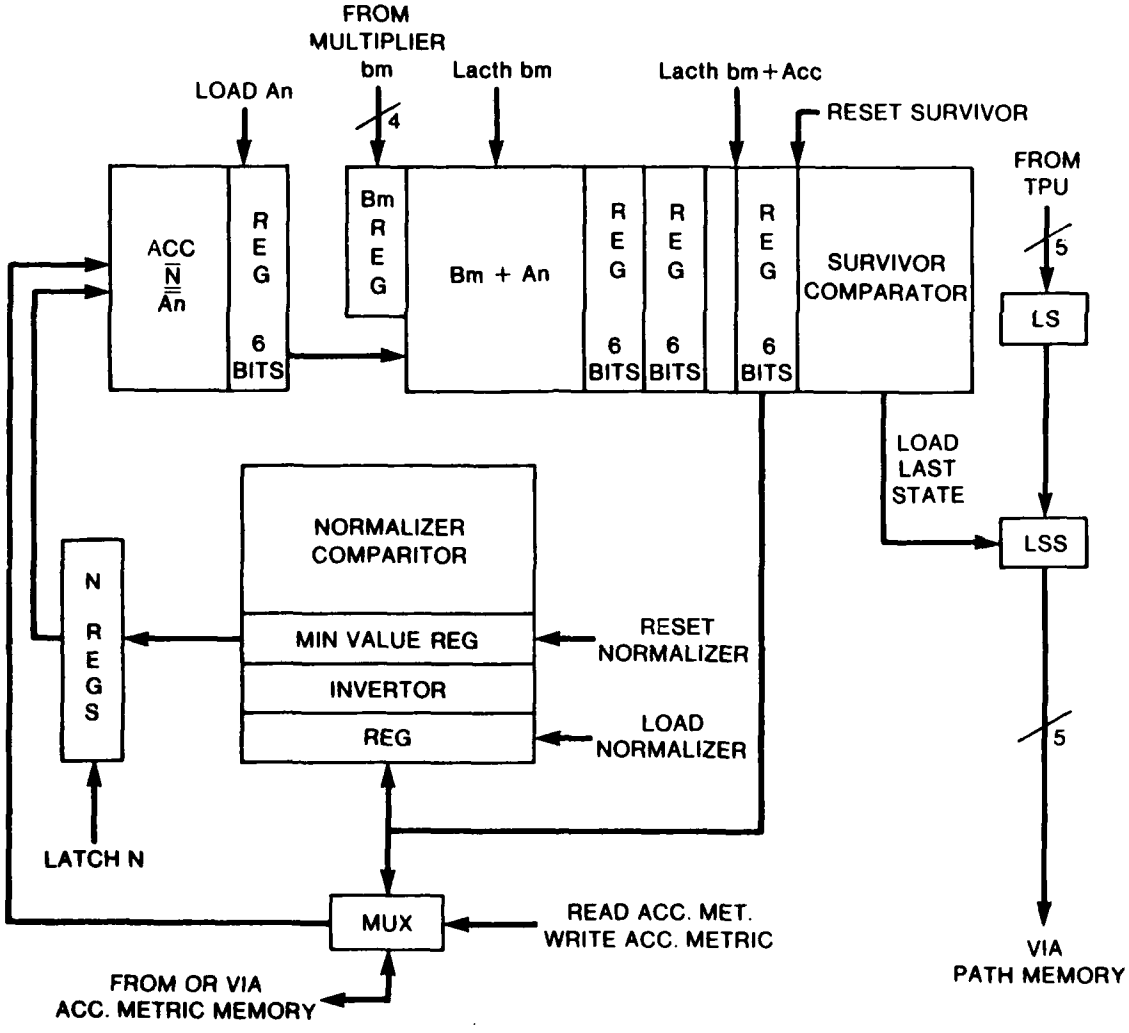
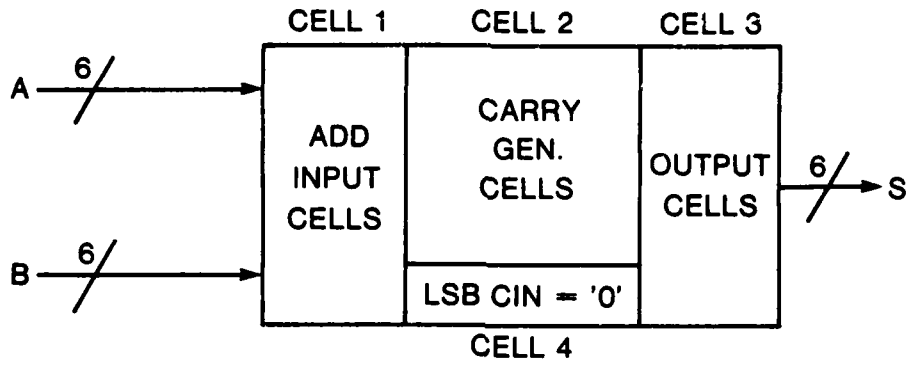


Figure 4.2. CPU

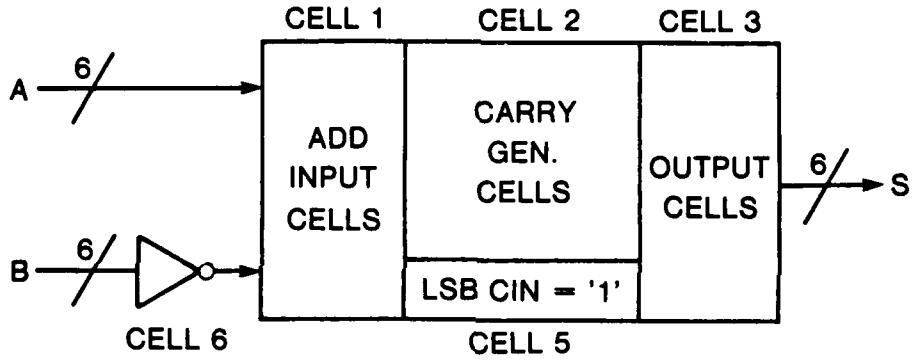
4 "FULL ADDER"

BLOCK DIAGRAM:



"SUBTRACTOR"

BLOCK DIAGRAM:



"COMPARATOR"

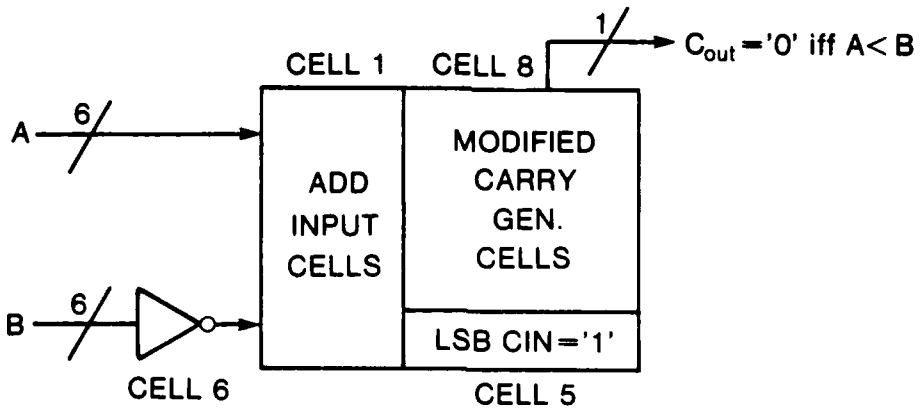
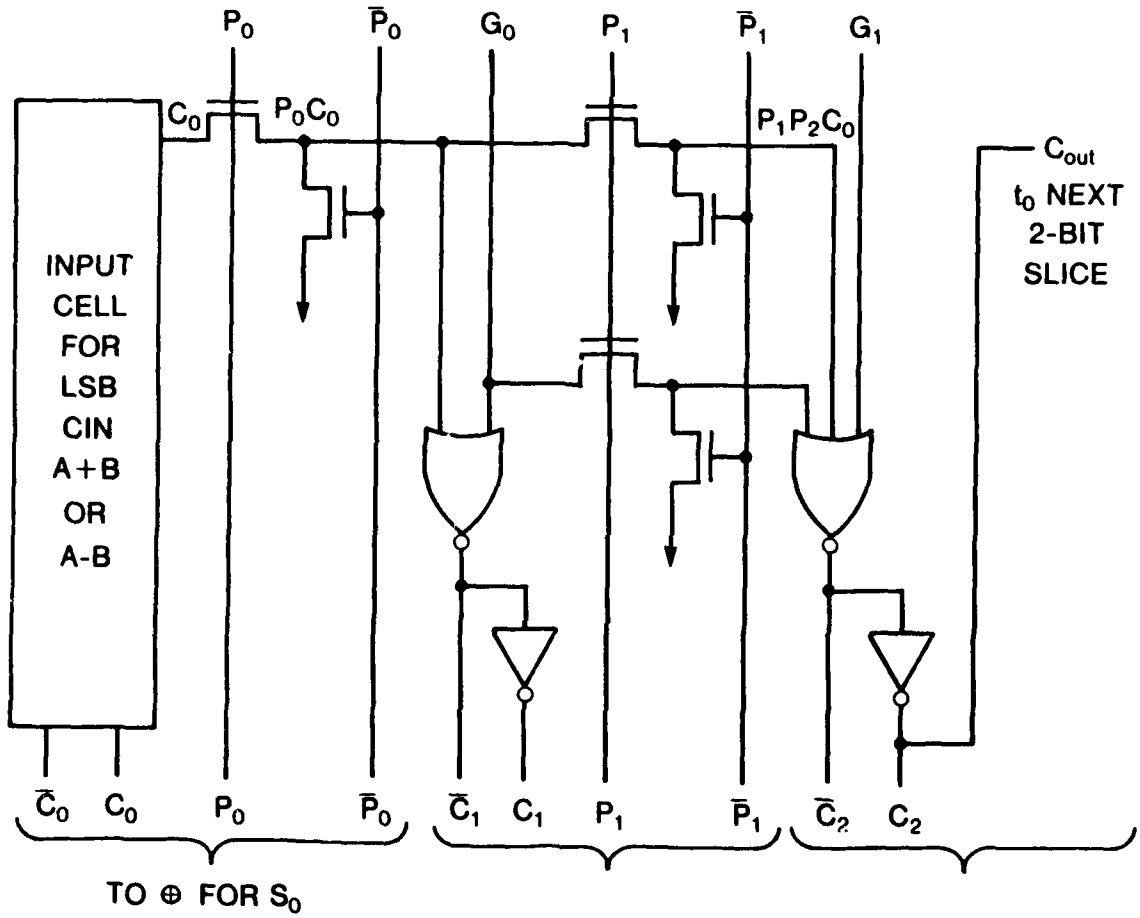


Figure 4.3. CPU Cells



$$C_1 = G_0 + P_0C_0$$
$$C_2 = G_1 + P_1G_0 + P_1P_0C_0$$

Figure 4.4. P,G,S Generation

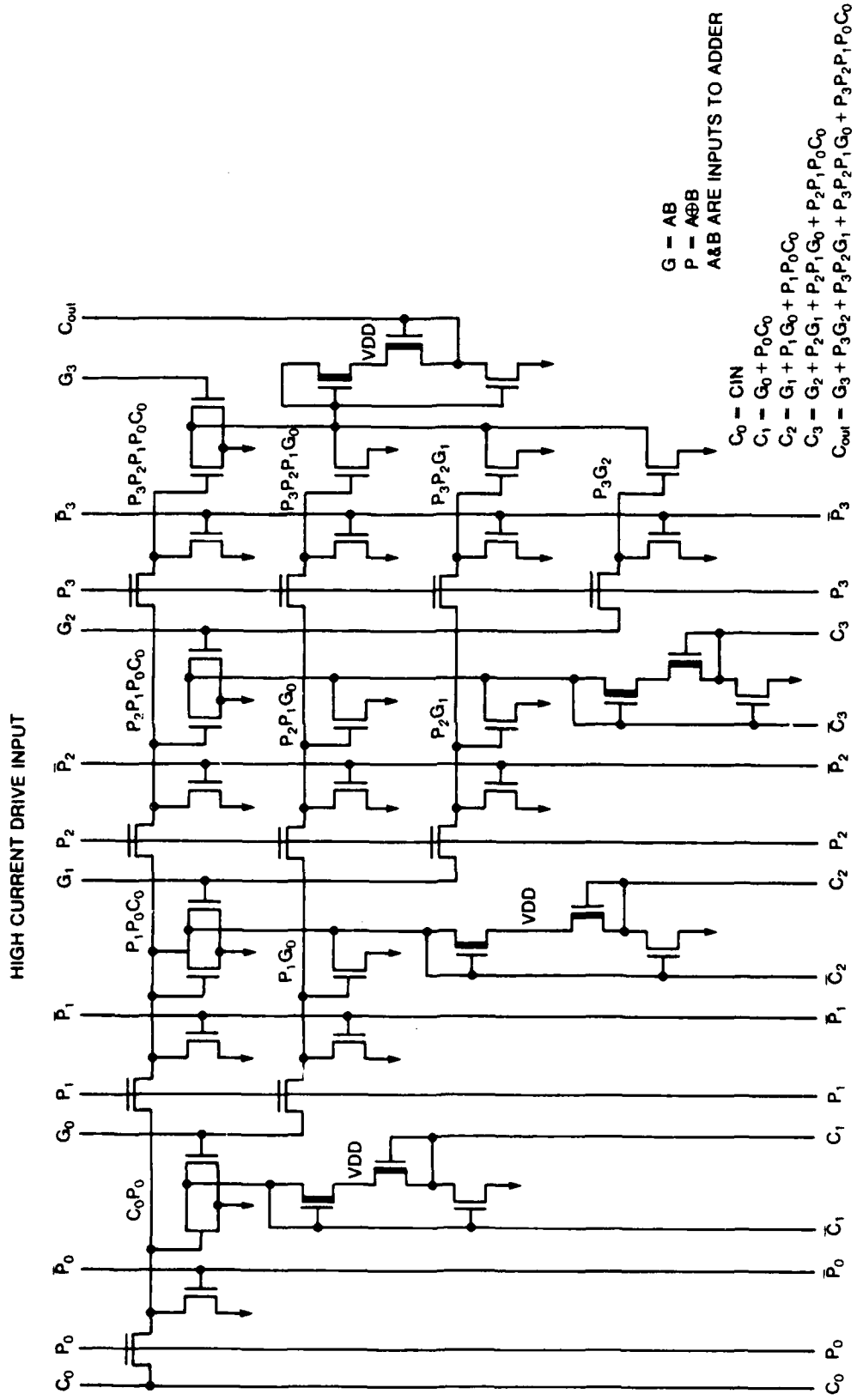


Figure 4.5. Carry Generation Logic Diagram

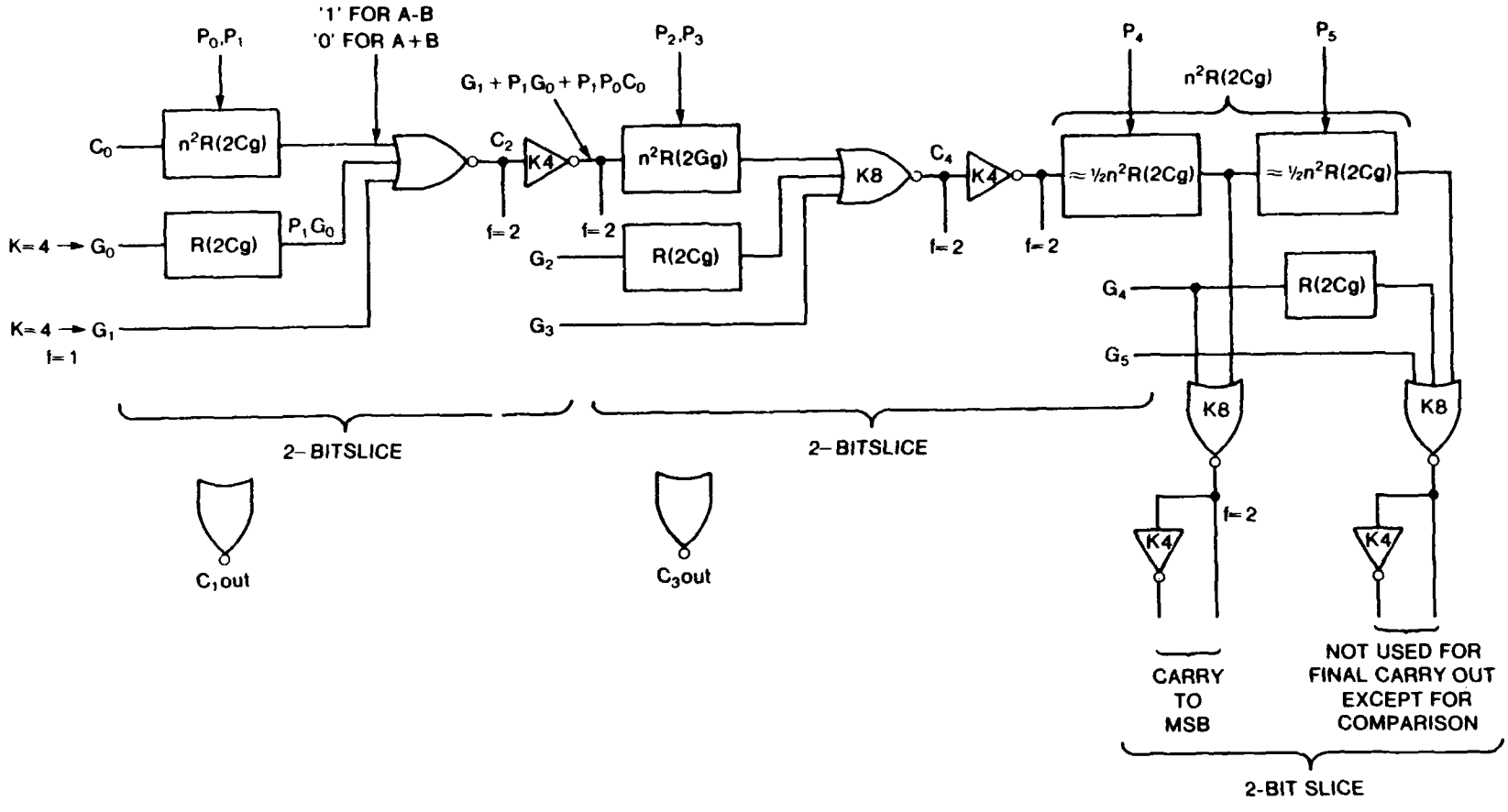


Figure 4.6. Circuit Diagram for Delay Analysis

CHAPTER 5
Trellis Processing Unit

Contributors:

Dan Asta
Alida Meinberg
Judith Chou

5.1 Project Description

The Trellis Processing Unit (TPU) is the subsystem of the Viterbi processor which generates the information necessary to address other subsystems*. The ouputs of the TPU are the values of I, LS, D, X1, X2. The only inputs to this sub-system are the control signals supplied by the controller. The role of this subsystem was discussed in Chapter 2.

In the actual implementation of the TPU, it is necessary to provide some delay in the path of the LS values in order to allow sufficient time for the multiplier to compute bm(1) & bm(0). The necessary delay is provided through an addition of a three word FIFO queue which is loaded as the LS values are generated.

When the system RESET occurs, the value of I and the queue are reset to all zeros. The counters used in this sub-system can also be used in a shift register mode for level-sensitive scan testing.

The block diagram for the TPU is shown in Figure 5.1.

* In a strict sense, the TPU could be regarded as part of the controller of the chip since it does not perform any function on the data path.

5.1 Implementation

The logic diagram of the TPU is shown in Figure 5.1

The considerations for the architecture of the TPU are: speed, layout simplicity and area consumed by the subsystem. The parent cells in this sub-system are:

1. Present State I Counter -- This is a static up counter using toggle flip-flops as its cells.

2. I $\oplus$ 3 PLA: A PLA accomplishes this addition; for the I+11 addition, the inversion of the MSB is strobed by an RS flip-flop.

3. Metric Coefficients X1, X2: These coefficients are coded in five bits with an additional bit reserved for sign. These are generated using a PLA.

4. Last State (LS) Counter: For each present state, LS(1)=I+3 is loaded into this counter, which is then run twice to generate LS(2) and LS(3). During the second period, the counter is loaded with LS(4)=I $\oplus$ 11 and then incremented twice. This counter is a parallel load counter.

5. The Queue: The queue is composed of 3 levels of master slave flip-flops for each LS and D bit; the queue is loaded synchronously when the LS counter becomes stable.

6. The Floor Plan: Was arrived at as a good compromise between efficient signal routing and efficient interface with other subsystems as shown in Figure 7.2.

5.3 Cells

These cells are discussed in detail in chapter 7.

a. D Flip-Flop, Master Slave: This cell is used as the building block of the queue.

b. Toggle Flip-Flop: This cell is used as the building block of all the counters in the TPU.

5.4 Timing and Simulation

1. I+3 PLA: Trise = 2.5 ns

2. X1, X2 PLA: Trise = 3.4 ns

3. Up Counters: Trise = 20 ns

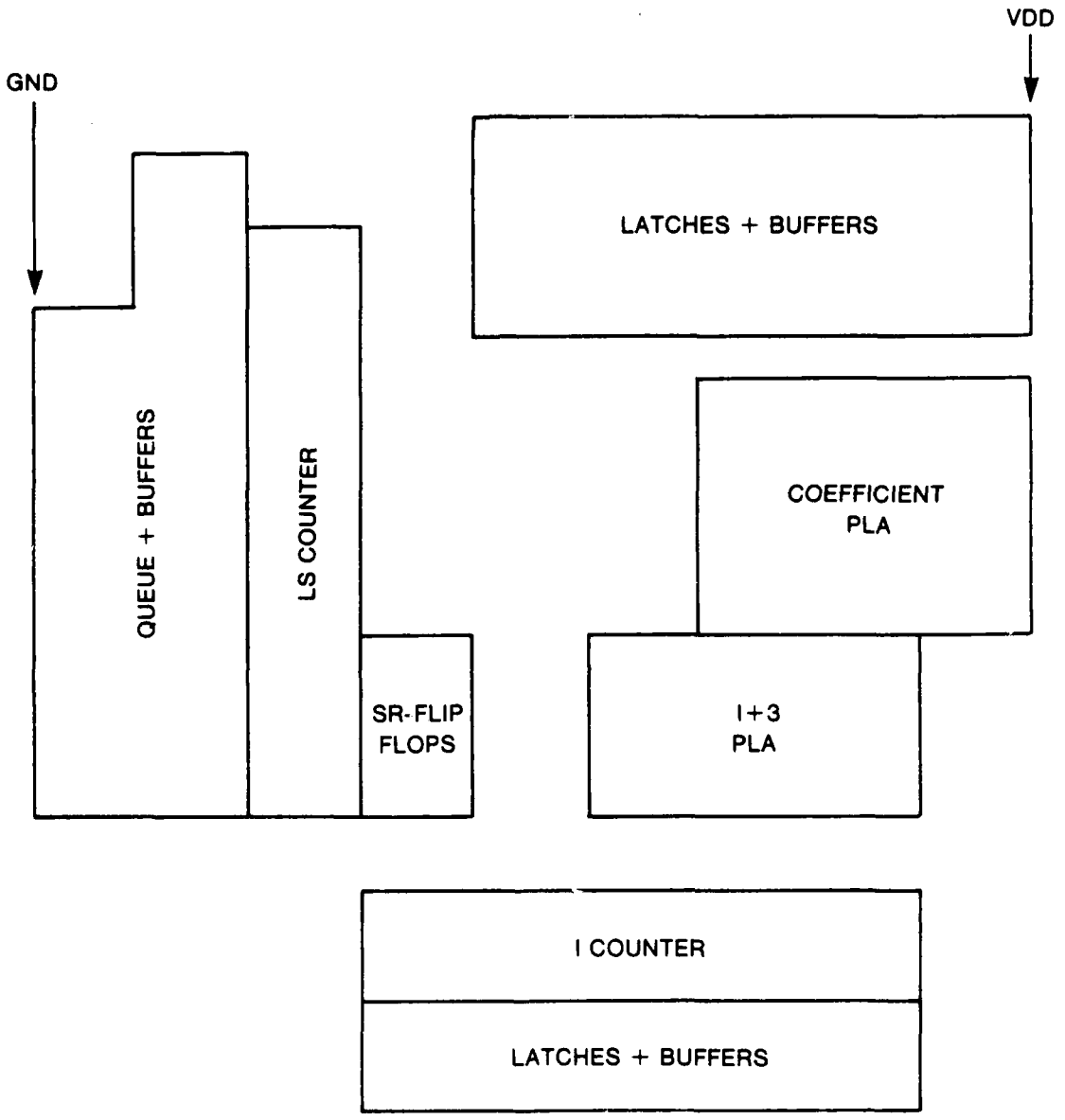


Figure 5.1. TPU

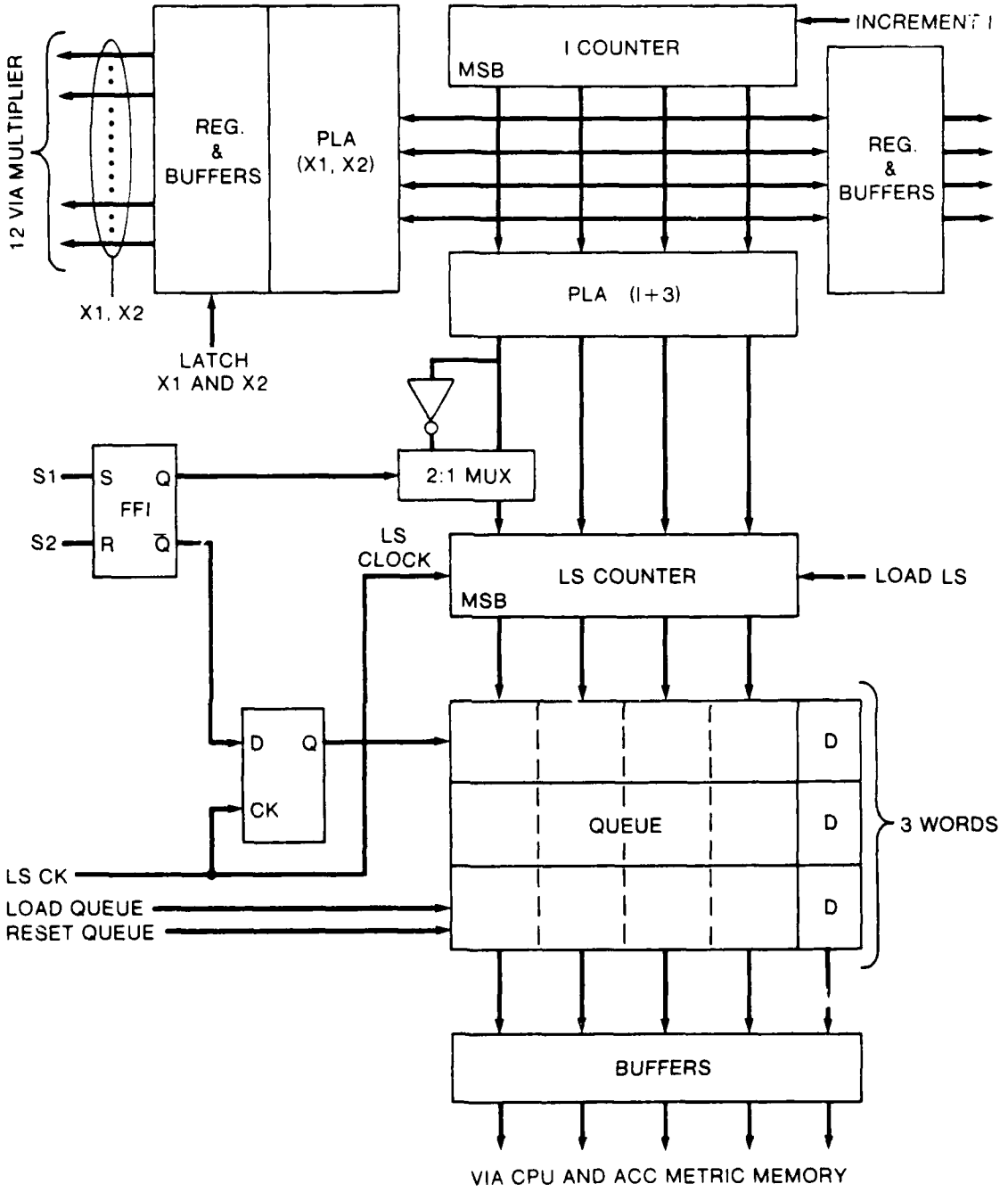


Figure 5.2. Trellis Processing Unit

CHAPTER 6
Memory

Contributors:

Bill Reber [1]
Steve Stillman [2]
James Bohannon

6.1 Project Description

The memory is composed of two independent blocks, namely, the Accumulated Metric Memory and the Path Memory. Nevertheless, the basic cells for both subsystems is the same.

Special features of the memory are the dual foreground/background parallel shift capability, global RESET and each cell is laid-out such that it can be used both as a source or destination within each column of the memory.

6.1.1 Accumulated Metric Memory

The CPU may store a survivor's accumulated metric value in the AM. The storage location is determined by the "state" of the system. The state is supplied to the AM on the "I" bus, which is four bits wide. Similarly, whenever the CPU needs to use the accumulated metric memory, the state is supplied by the LS bus.

6.1.2 Path Memory

The TPU generates the output bit for each of the sixteen states. Each time one is generated, the path of the preceding survivor state is copied to to the path of the present state (source path selected by the LSS bus, destination path selected by the I bus). The present path is then shifted one bit, and the new bit is appended. The convention is that bit #0 is the new bit, and #29 is the oldest. During the shift, the high bit is discarded.

---

[1] Path Memory

[2] Accumulated Metric Memory

6.2 Implementation

To accomplish selection of registers, a 4-to-16 *decoder* is used. The decoder should also have inputs to select all or none of the registers.

I. *PM Decoder* -- For the path memory, one decoder is used for the front registers, and an additional one is required for the back-ups.

II. *AM Decoder* -- The accumulated metric memory only needs one decoder with extra logic to connect to either the main registers or the back-ups. This can be done since the registers will never be selected at the same time. For this reason, inputs/outputs can be accomplished on the same bus.

The paralleled transfer of the back-ups to the main registers is facilitated by connecting the front and back-up register cell to the same bus. The basic *memory cell* contains two bits, one for the main and the other for the back-up.

It turns out that the operations of copying one path to another, shifting the new path and inserting the input bit can be combined into a single operation. This is done by selecting source and destination path registers at the same time. Since copying takes place only from a main register to a back-up register, and not vice-versa, then, by simply connecting the output of each main register bit to the "right hand bus" and connecting the input of the back-up register bit to the "left hand bus," the shift is accomplished at the same time as the copy. The input bit DIN is connected to the left-most bus.

The timing diagram as shown in Figure 6.3 depicts the relative timing of the control signals. The duration of each pulse can be increased, depending on the system clock frequency. All registers are static.

6.3 Cells

I. Register Cell -- The basic register cell (burreg) used in the memory was designed using buried contact. The size of the burreg is 55 λ x 56 λ *. This resulted in 34% reduction in the area of the entire memory as opposed to using the butting contact. The circuit and transistor models for the basic register cell are shown in Figures 6.6.b, 6.6.c.

II. Decoder -- The decoders are designed with the use of simple pass transistor logic. Control lines run vertically, and output select lines run horizontally. The transistor model for the decoder is shown in Figure 6.6.a

III. The overall floor plan is shown in Figure 6.4.

The design team originally designed the butting contact version, and its size was 122 λ x 71 λ.

6.4 Timing and Simulation

There will be certain set-up times involved for events taking place in the memory. The following names will be defined to describe the delays:

Path Memory

1. PDst - decoder set-up time

2. PMst - memory array stabilization time

Accumulated metric memory

1. Adst - decoder set-up time

2. AMist - memory array stabilization time

3. AMost - memory array stabilization time for the output

Both Memories

1. XFRst - length of XFR required

2. RSTst - length of RST required

The timing delays are:

PDst 92 ns
PMst 228 ns
ADst 117 ns
AMst 152 ns
XFRst 768 ns
RSTst 768 ns

The various circuit models used for these estimates are shown in Figure 6.4.

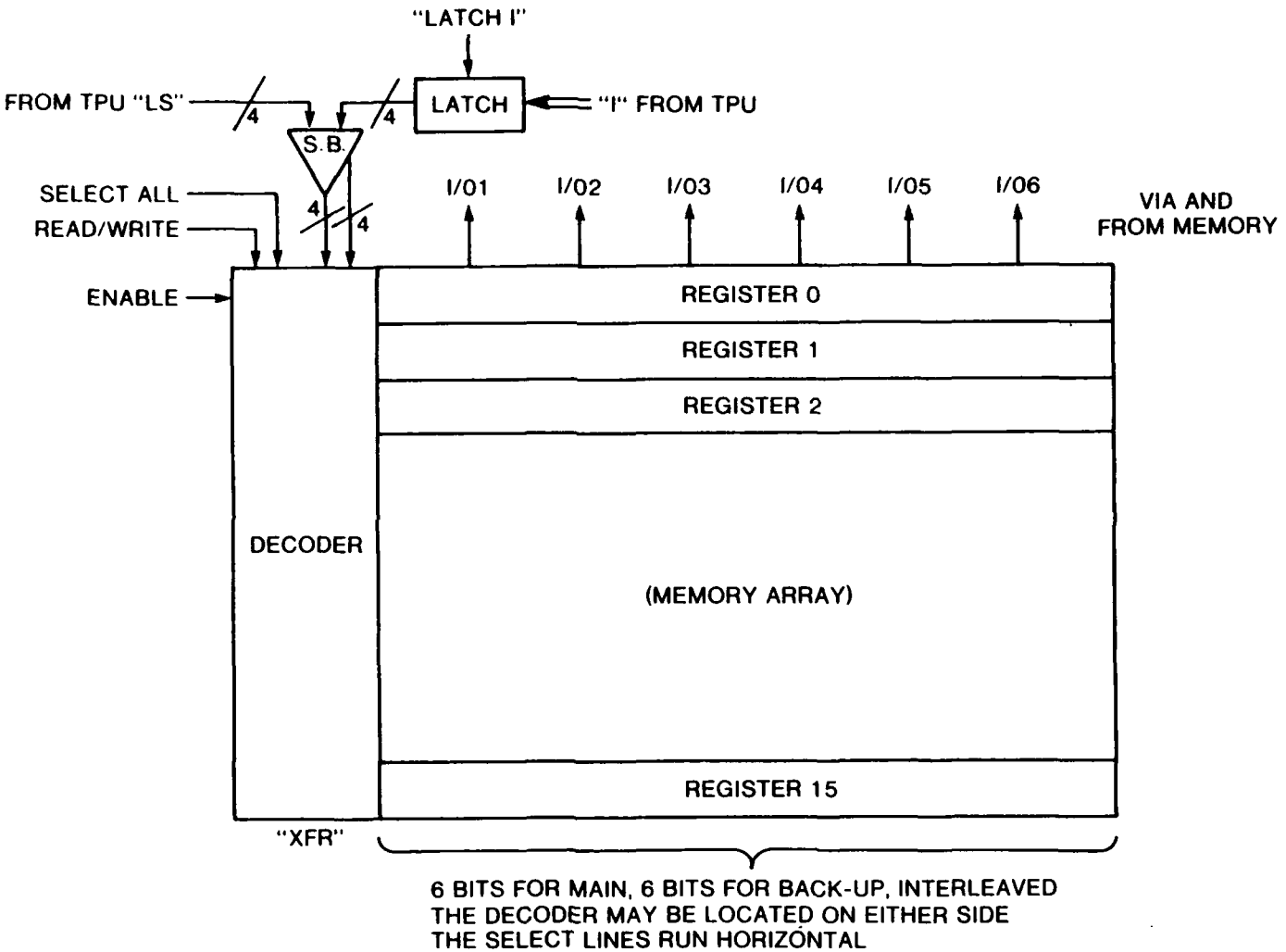


Figure 6.1. Accumulated Metric Memory

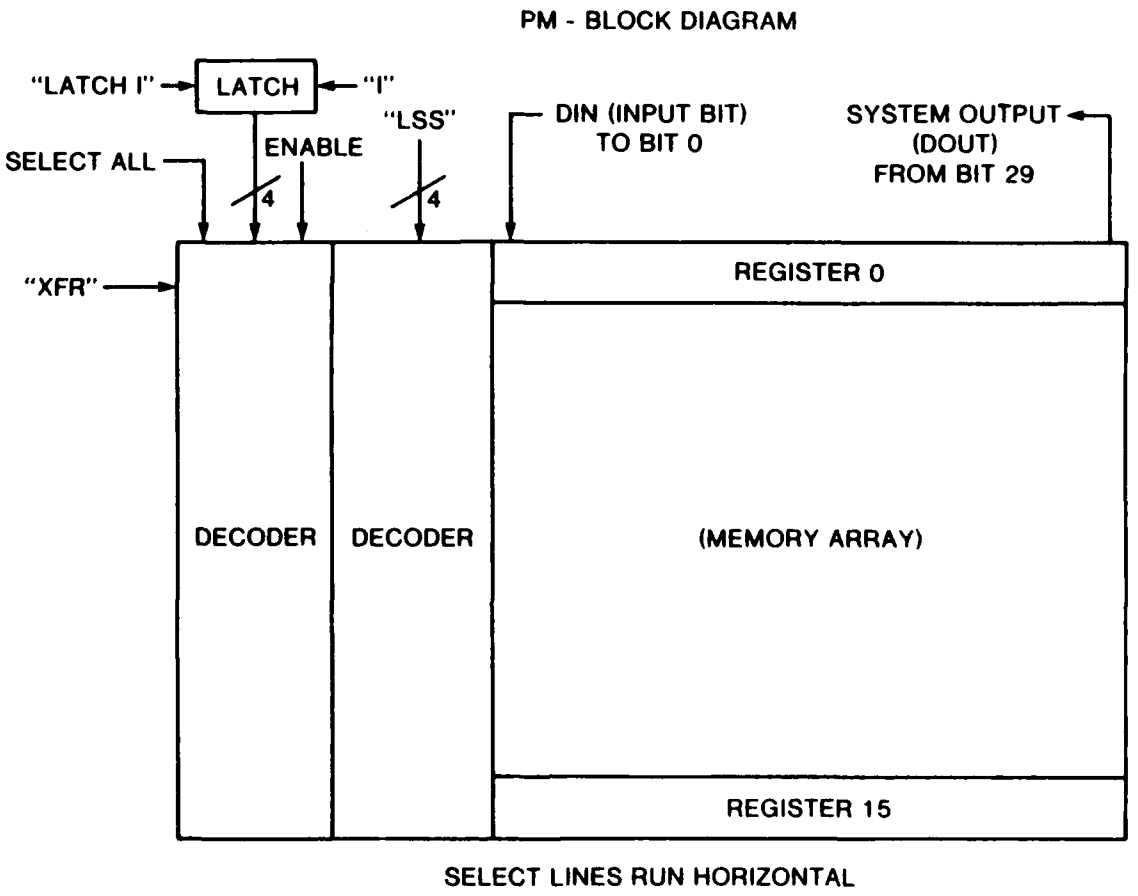
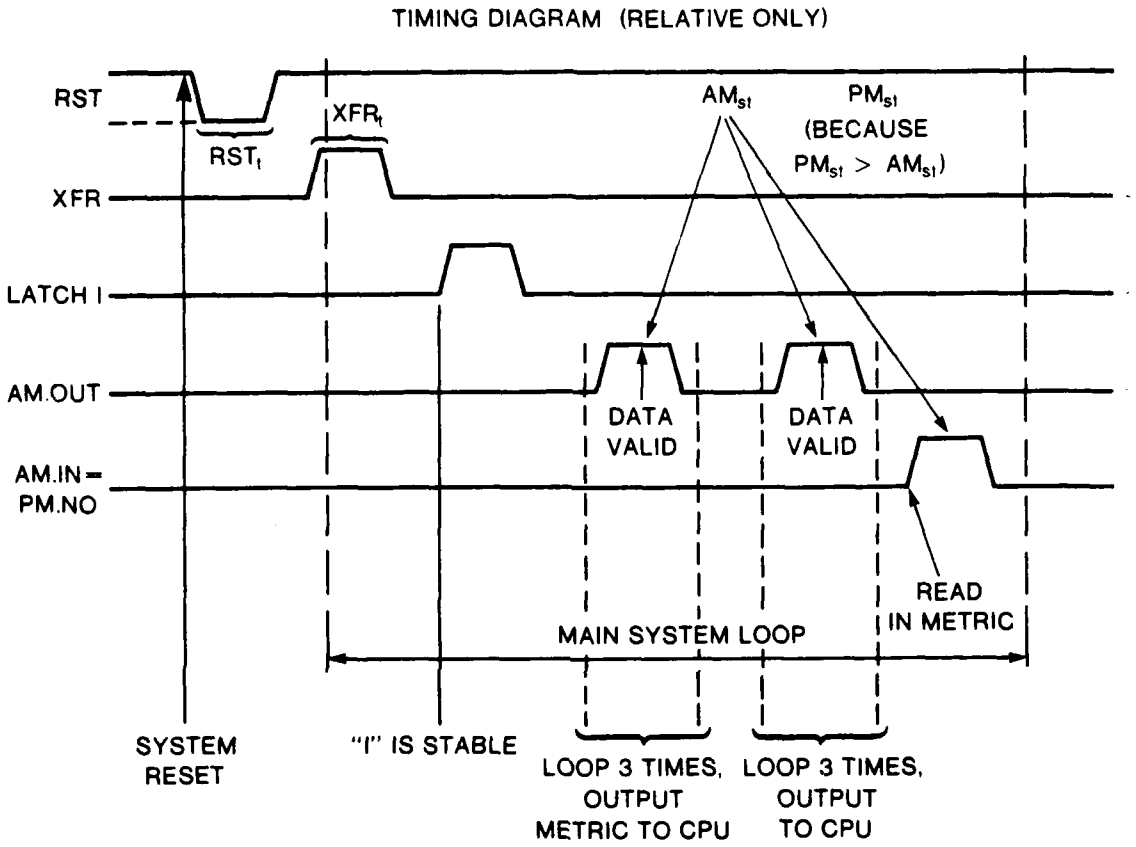


Figure 6.2. Path Memory



(1) XFR is the same for AM and PM.

(2) "LS" and "LSS" must be stable a certain amount of time before raising AM.IN or AM.OUT to allow the decoders to set up.

Figure 6.3. Memory Control Signals

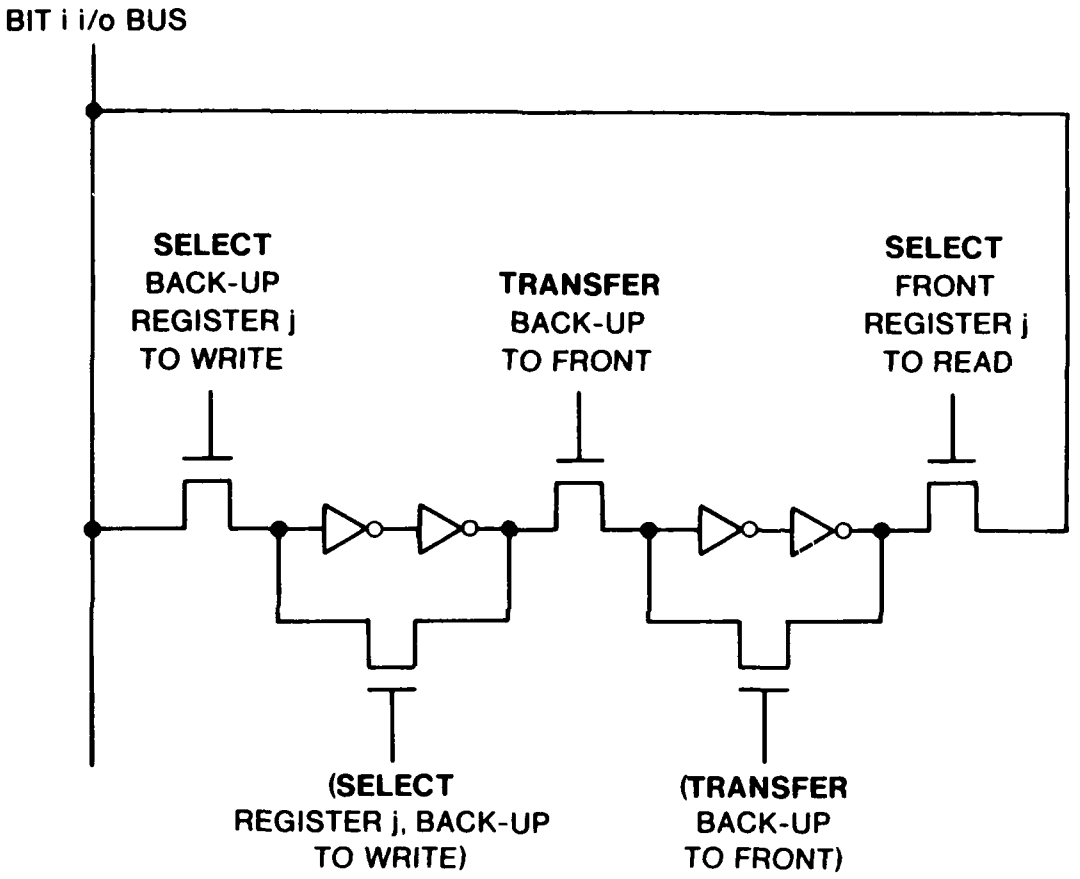


Figure 6.4 Logic Diagram for an Accumulated Metric Register Cell

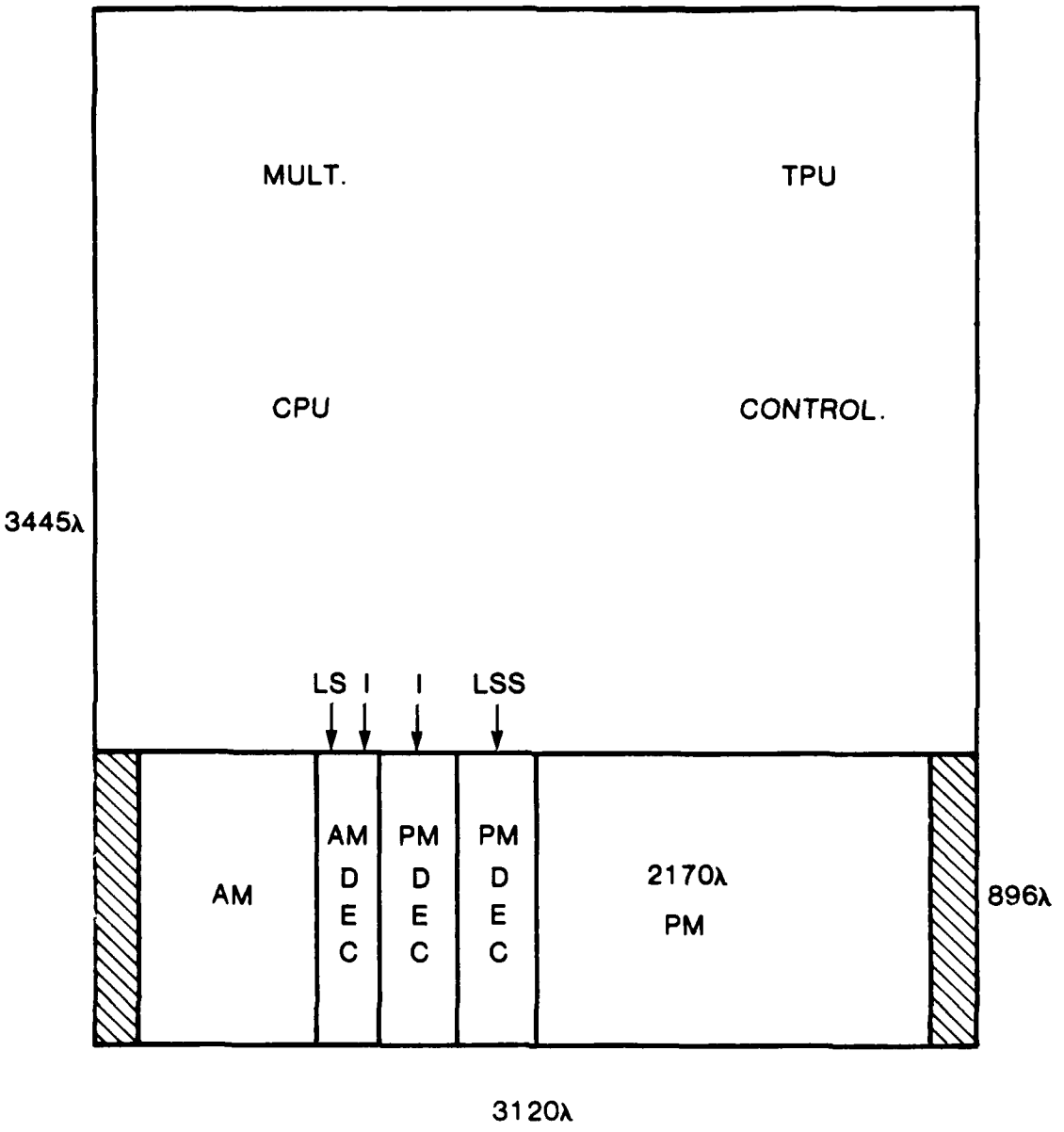


Figure 6.5. MEMORY FLOOR PLAN

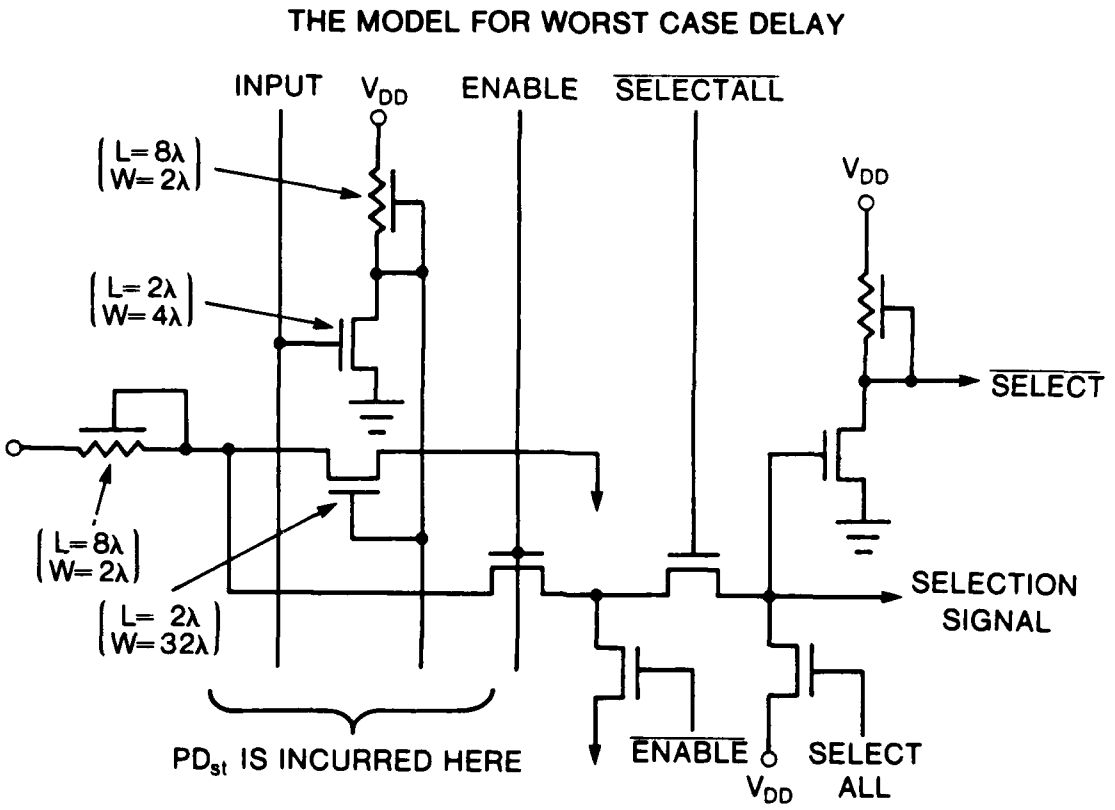


Figure 6.6.a. Decoder Transistor Model

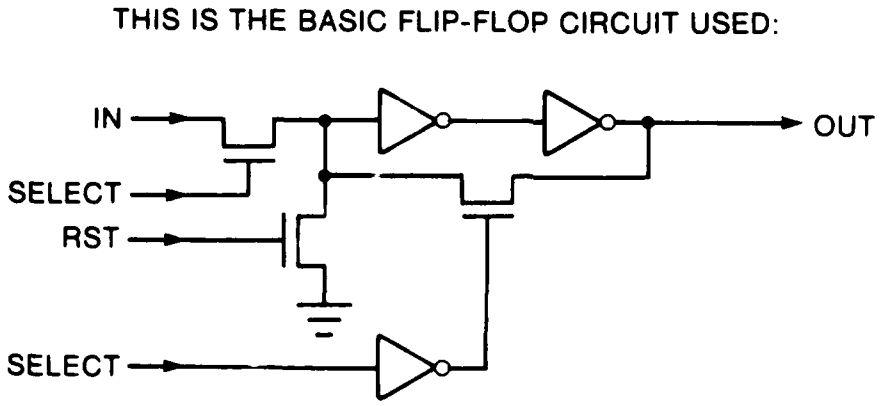


Figure 6.6.b. Path Memory Register

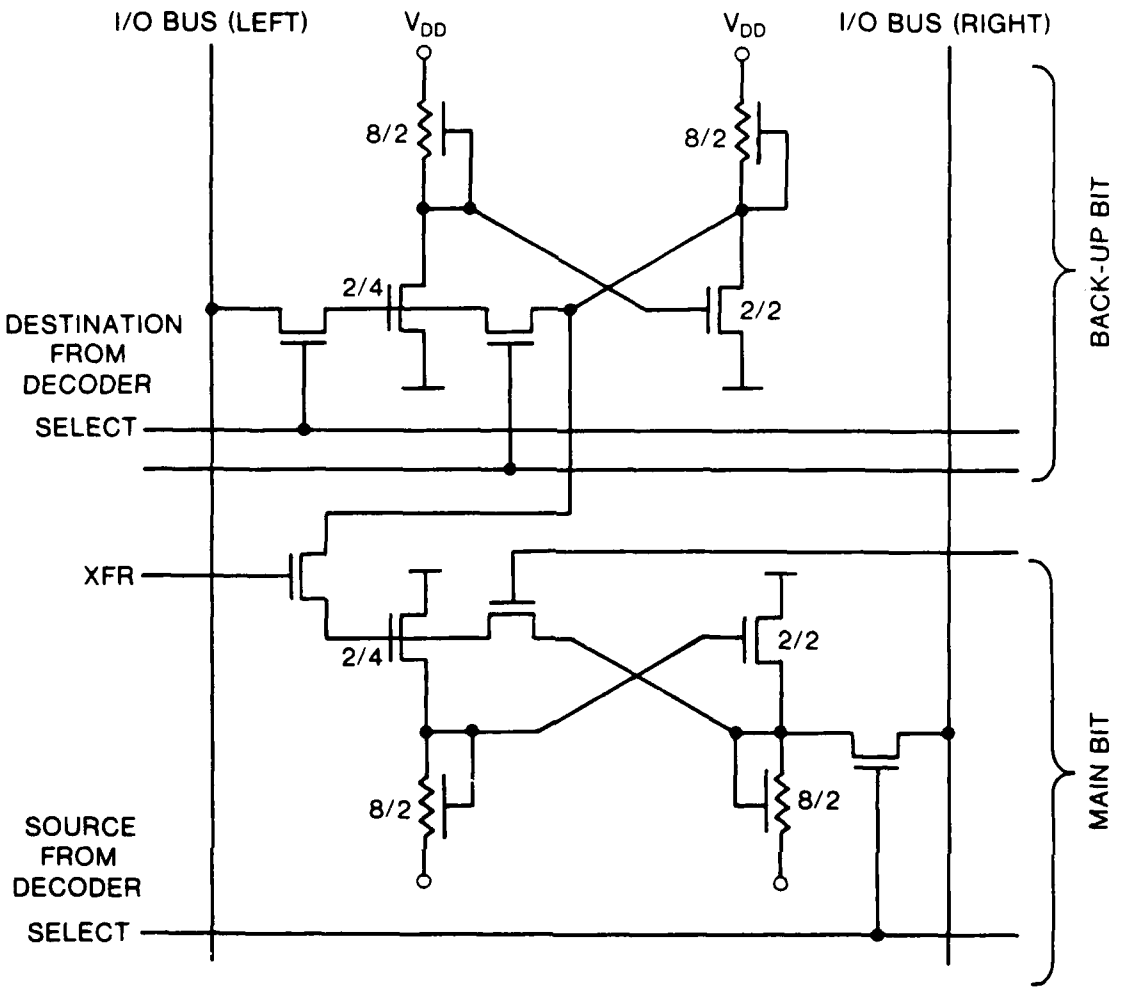


Figure 6.6.c. Transistor Model for Basic Register Call

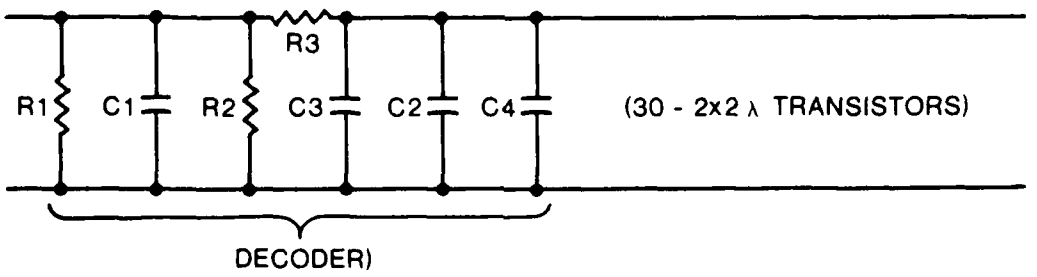


Figure 6.6.d. Circuit Model for Delay Analysis

CHAPTER 7
Controller
Contributors:

Ramin Sadr
Wade Mergenthal*
Steve Yinger
Joe Jensen

7.1 PROJECT DESCRIPTION

The system structure of the controller was discussed in chapter 2. At every clock cycle, the controller sequences through a memory and outputs a word of information. The bits in this word are used as control signals to functions in other parts of the chip. The control signals are to be used, not as a two-phase clock to gate latches, but as a window during which a particular function is to occur. The controller itself operates on a two-phase non-overlapping clock and outputs a new set of control signals on the rising edge of phi 2. These outputs remain valid until the next phi 2. One clock cycle is the period for one phi 1 and phi 2.

The most direct approach to implementing the microprogram is to store the instructions in sequential order in a memory. To sequence through the instructions, we need a program counter which increments every clock cycle and provides the next highest address. However, in the Viterbi algorithm, many of the calculations have to be done over and over again; so, it would be advantageous to implement some of the instructions in loops. Two kinds of loops are needed, one that is done 16 times and the other 3.

7.2 Implementation

The block diagram for the controller is shown in Figure (7.1).

Control signals sent to other parts of the chip go directly into superbuffers specially designed to boost their drive capability. The remaining seven bits are used by the controller itself. Two of these bits, called t0 and t1, are used together to indicate the end of a loop and identification of the particular loop ended. If t0 and t1 both equal zero, then we are in the middle of the loop and want only to increment the program counter and continue on to the next instruction. If, however, we are at the end of a loop, then t0 and t1 indicate such. Also connected to the loop PLA are a divide-by-three counter and a

* Special thanks are due to Mr. Mergenthal for his work on this subsystem.

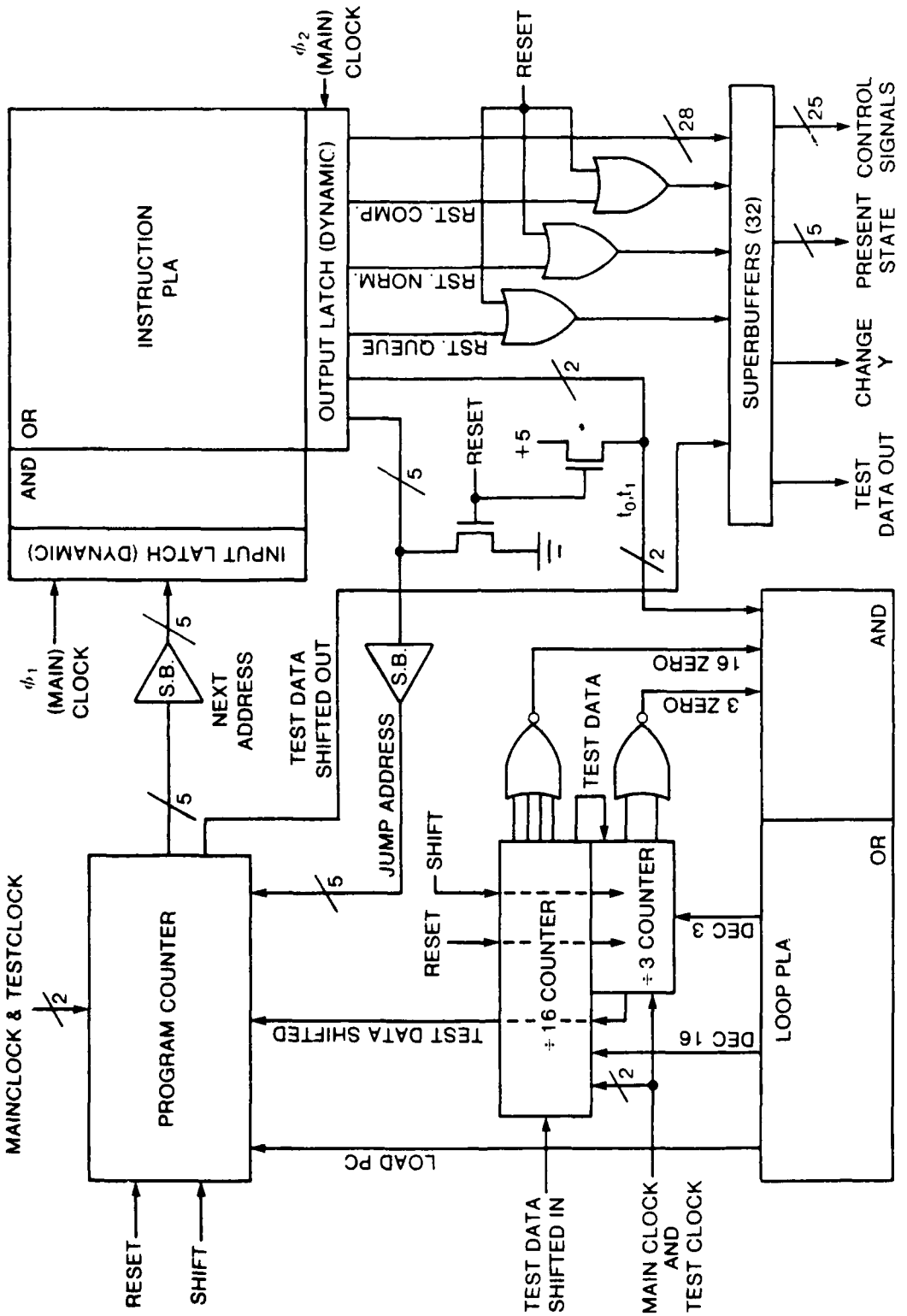


Figure 7.1. Controller Block Diagram

divide-by-sixteen counter. These are used to indicate how many times the respective loop has been executed. They are both down-counters and will have all zeros in them when the loop has been executed the proper number of times. When the end of a loop is indicated by t0 and t1, the loop PLA looks at the appropriate counter and decides if it has been executed the proper number of times. If it has, the loop PLA allows the program counter to be incremented to exit from the loop and decrements the appropriate loop counter. If the loop has not been executed enough times, the loop PLA sends a signal to load the address of the beginning of the loop into the program counter. It also decrements the appropriate loop counter. The address of the beginning of the loop is provided from the instruction PLA as the other five control signals to the controller from the instruction word.

It takes two full clock periods for information to be cycled through the controller. As one instruction is being output, the address of the next instruction is being output from the program counter. At the same time, the address of the next instruction after that is being determined by the loop PLA. This means that t0 and t1 must indicate the end of a loop, not in the last instruction in the loop but in the next to last instruction in the loop. The jump address of the beginning of the loop must also be in this same instruction. This means that the minimum loop size is two instructions. This is indicated in Figure 7.2, which gives the contents of the instruction PLA.

7.2.1 Level-Sensitive Scan Design

The controller was designed using the level-sensitive design approach as outlined in the paper "A Logic Design Structure for LSI Testability" by E. B. Eichelberger and T. W. Williams [12]. Their approach required that all internal storage elements be designed to operate as shift registers as well as normal storage nodes. All our static latches were designed with shift register capabilities. The second requirement was that the design be level-sensitive (LST).

The program counter in the controller is modified such that it can also be used as a shift register for LST testing; for this purpose the main clock is frozen and a test clock is introduced to shift in/out the p.c content via an output pad. Both clocks are inverted at the clock pads then or'ed below th -16 counter.

7.2.2 The Control Signals

The control signals generated by the instruction PLA are shown in Figure 7.3.

The controller, as well as parts of the rest of the chip, must be initialized upon power-up. A global RESET pulse is provided for from off-chip. In order to be effective, it must last for at least 3 full clock periods. The signal then gates transistors in the controller to force the t0 and t1 outputs high, the jump address lines low and to reset the 16-counter, 3-counter, and 30-counter. This forces the first instruction, T0, to be output from the controller, which contains two signals to the CPU for initialization. The T0 instruction will remain valid for two clock periods after the reset line goes low. On the third clock cycle, the next instruction, T1, will be executed, which contains one more reset signal for

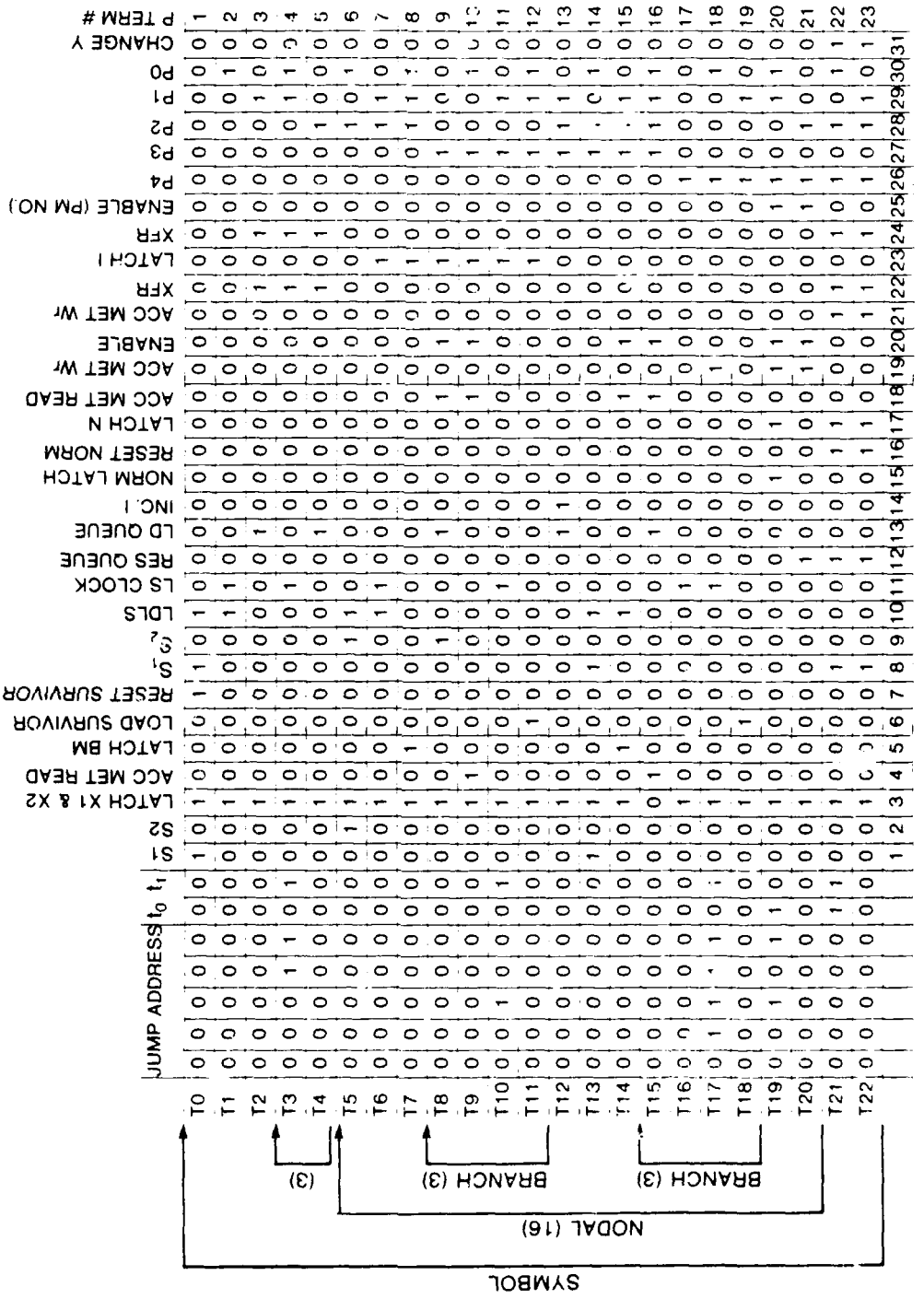


Figure 7.2. The Instruction PLA Truth Table

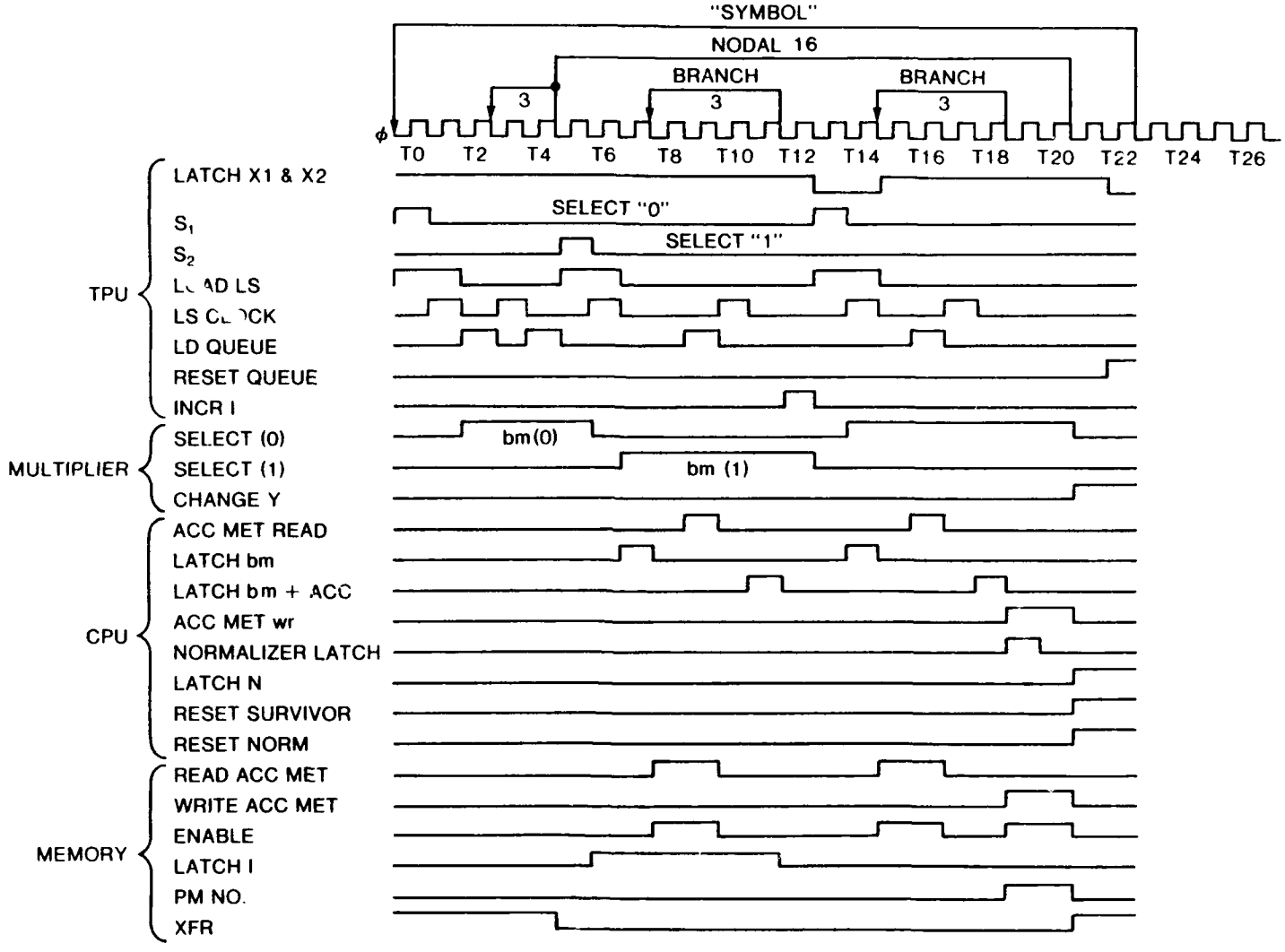


Figure 7.3. Control Signals

the CPU. After this, the controller cycles in its normal manner to instruction T2, etc.

Following are descriptions and designs for each of the components in the block diagram:

7.3 Cells

I. *Instruction PLA*

The instruction PLA is a PLA made through use of the "mkfsm" program having 5 inputs, 26 p-terms and 34 outputs. The inputs are clocked by phi 1, and the outputs are clocked by phi 2. Each row of the PLA contains one instruction word of the microprogram corresponding to the system timing diagram shown in Figure 7.3. The contents of the PLA are shown in Figure 7.2. The instruction words are labelled from T0 through T25, each corresponding to one clock cycle on the system timing diagram. Loops are indicated by lines on the left side of Figure 7.2. The p-term number associated with each word is on the right. This PLA is 396 *lamda* x 286 λ and has 5 inputs, 32 outputs and 22 p-terms.

II. *Loop PLA*

The loop PLA is used to determine what is done when the end of a loop is reached. It has four inputs (t0, t1, 16-counter zero and 3-counter zero), three outputs (decrement-16 counter, decrement 3counter and load program counter) and seven p-terms. The inputs and outputs are not clocked directly at the PLA. This PLA was made with "mkfsm", and its truth table is shown in Figure 7.4. This PLA is 132 λ x 132 λ and has 4 inputs 3 outputs and 7 p-terms.

III. *Counters*

As has already been mentioned, there are four counters in the controller, one of which has not yet been implemented. All are synchronous counters, and all are built using a modified version of the TPU group's toggle flip-flop as the basic building block. Following is a focus on that basic cell and the design method used to construct the counters. Afterwards, each counter will be discussed individually.

Divide-by-3 counter is a 2-bit down-counter used to count the number of iterations of the branch loop (Recall that this computes the new accumulated metric for a transition.). The Reset line shown on the block deagram will be connected to the LOAD input, and DIN is set so that, after 3 passes through the loop, the counter state will be 00. For this counter, DIN(0) is tied to GND, and DIN(1) is tied to +5 Volts. The Reset need only occur once; after the 00 state, the counter is set to return to the 10 state. We want phi 1 and phi 2 to decrement the counter only at the end of an iteration of the appropriate loop; so, when we do not want to decrement, a pass transistor to GND and ENbar is provided to ground T. The counter will be allowed to decrement only when ENbar (shown on block diagram as Decrement 16bar) is low.

	0	0	0	0	1	1	0
	0	0	0	1	1	1	0
INCREMENT	0	0	1	0	1	1	0
ALWAYS	0	0	1	1	1	1	0
	0	1	0	0	1	0	0
NODAL	0	1	0	1	1	0	0
CYCLE	0	1	1	0	1	0	1
	0	1	1	1	1	0	1
	1	0	0	0	0	1	0
BRANCH	1	0	0	1	0	1	1
CYCLE	1	0	1	0	0	1	0
	1	0	1	1	0	1	1
	1	1	0	0	1	1	1
SYMBOL	1	1	0	1	1	1	1
CYCLE	1	1	1	0	1	1	1
	1	1	1	1	1	1	1
INPUT # →	1	2	3	4	1	2	3 ← OUTPUT #

p term #	p term
1	$\overline{t_1}$
2	t_0
3	$\overline{t_0}$
4	t_1
5	$t_1 \cdot t_0$
6	$t_1 \cdot 3zero$
7	$t_0 \cdot 16zero$

$\overline{DECREMENT3} = \overline{t_1} + t_0$

$\overline{DECREMENT16} = \overline{t_0} + t_1$

LOAD PC $= (t_1 \cdot t_0) + (t_1 \cdot 3zero) + (t_0 \cdot 16zero)$

Figure 7.4. Loop PLA Truth Table

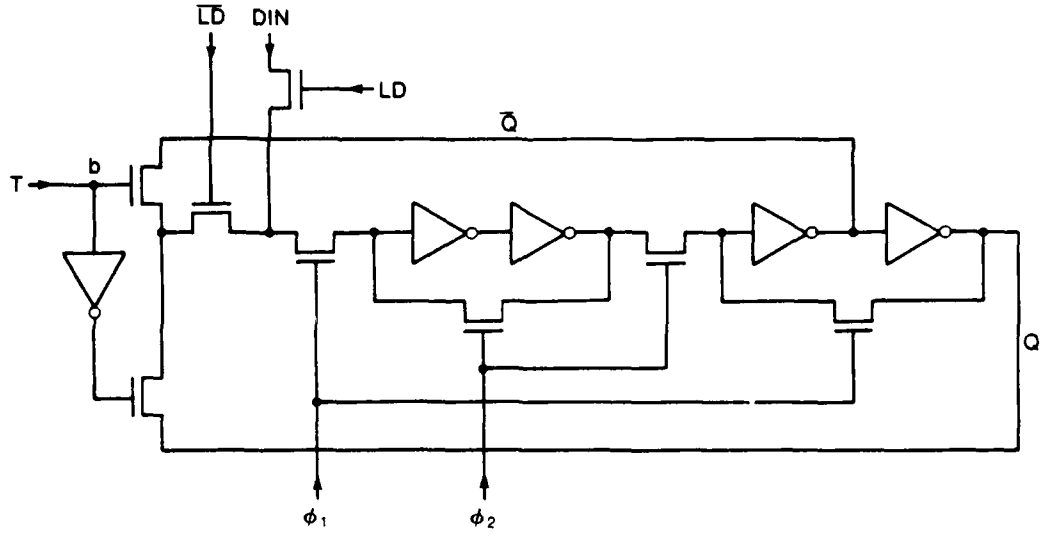


Figure 7.5. TPU Group's Toggle Flip-Flop

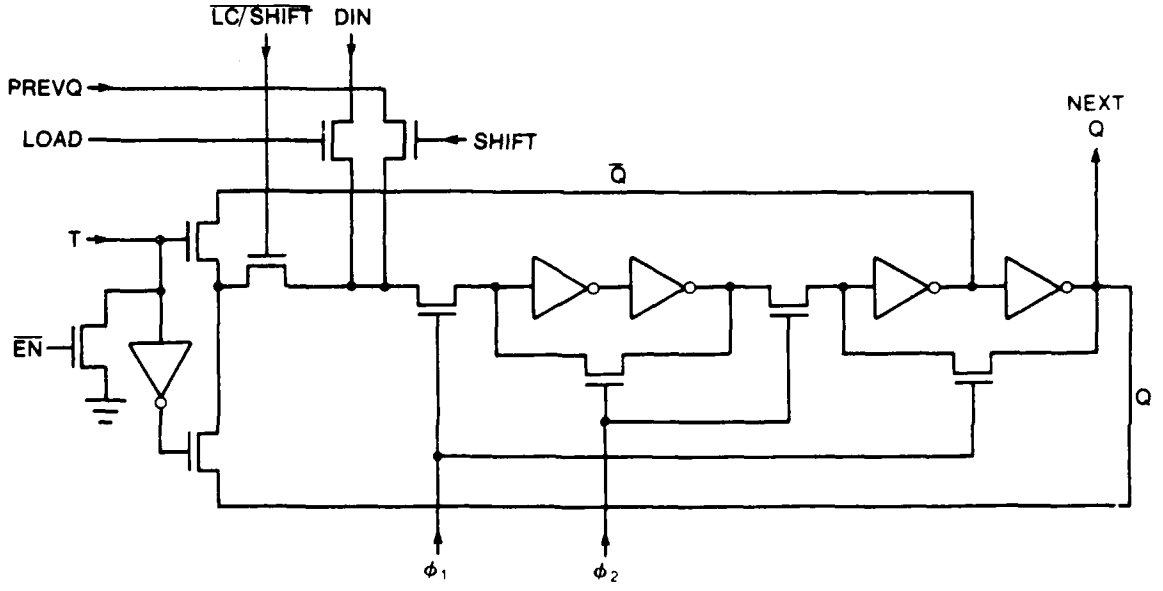


Figure 7.6. Flip-Flop Modified to Include Shifting Capability

To indicate to the loop PLA when the count has reached zero (i.e., the proper number of loop iterations completed), the counter outputs are NORed together; the signal 3zero goes high at that time. The transition table and counter block diagram are shown in Figure 7.7. Its size is 240 λ by 86 λ.

Divide-by-16 counter is a 4-bit down counter used to count the number of times the nodal loop is executed (once for each of the 16 states in the trellis). As with the 3-counter, the Reset line will connect to LOAD and will set the counter to a predetermined value -- for this counter, 1111; therefore, for each cell, DIN is tied high. ENbar has the same function as in the 3-counter, as does the NOR gate, which has 4 inputs for this counter. On the 16th iteration, 16zero goes high, and the loop PLA will not allow another iteration. The next state will again be 1111, ready for another series of iterations. The transition table, Karnaugh maps and block diagram appear in Figure 7.8. The size of this counter is 495 λ x 96 λ.

IV. *Program counter*

This is a 5-bit up-counter which provides the address of the desired instruction word to the instruction PLA. The two major differences between it and the 3- and 16-counters are: 1. The DIN inputs are not tied high or low but are connected to the loop-beginning-address field of the instruction word. This allows a jump to occur when the loop PLA instructs. 2. The ENbar feature is not incorporated in this counter because there is never a time when we do not want to increment. We want to run continuously on phi 1 and phi 2. Also, instead of Reset, LOAD will be connected to the loop PLA -- instruction word 00 contains the initialization information for this counter. The flip flop cells for this counter are modified to allow shifting for level-sensitive scan testing. The transition table, Karnaugh maps, and block diagram appear in Figure 7.9. The size of this counter is 552 λ x 99 λ.

V. *Buffers*

The Superbuffers are designed to handle the current requirement to derive the control lines. From the chip layout, it was determined that the longest path among the control busses is 2200 λ. Assuming five loads per line, the capacitance was estimated at .83 pFarad or an equivalent of 130 loads. The logic diagram of the superbuffer and its transistor model are shown in Figure 7.10. This cell was also laid out to fit almost within the pitch of the output control lines from the instruction PLA.

7.4 Timing and Simulations

A level-sensitive design is not dependent on risetime, falltime, minimum delay of the individual circuits or wire delays. The only dependence is that the total delay through a number of levels be less than some value.

Divide-by-3 down-counter

TRANSITION TABLE:

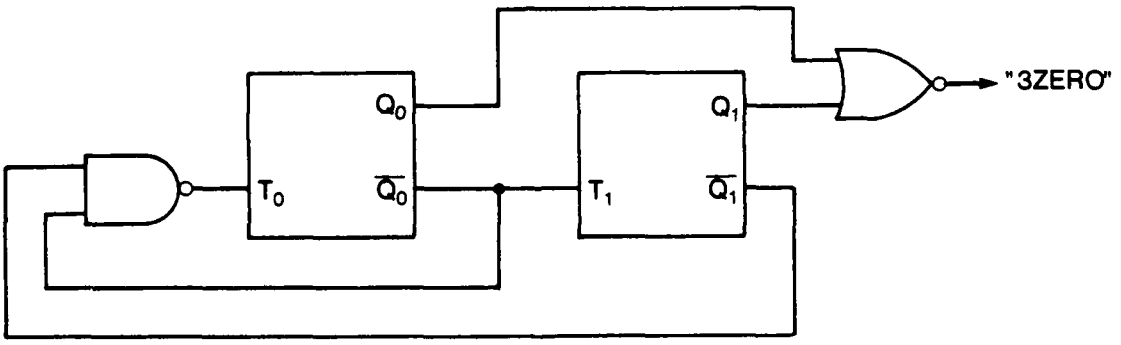
Input		Present State		Next State	
T_1	T_0	Q_1	Q_0	Q_1	Q_0
1	1	1	0	0	1
0	1	0	1	0	0
1	0	0	0	1	0

ON RESET, SET PRESENT STATE TO 10

$$T_0 = Q_0 + Q_1 = \overline{\overline{Q_0} \cdot \overline{Q_1}}$$

$$T_1 = \overline{Q_0}$$

THIS LEADS TO:



ALSO, BOTH CELLS GET:
LOAD, SHIFT, $\phi_1, \phi_2, \overline{LD/SHIFT}$

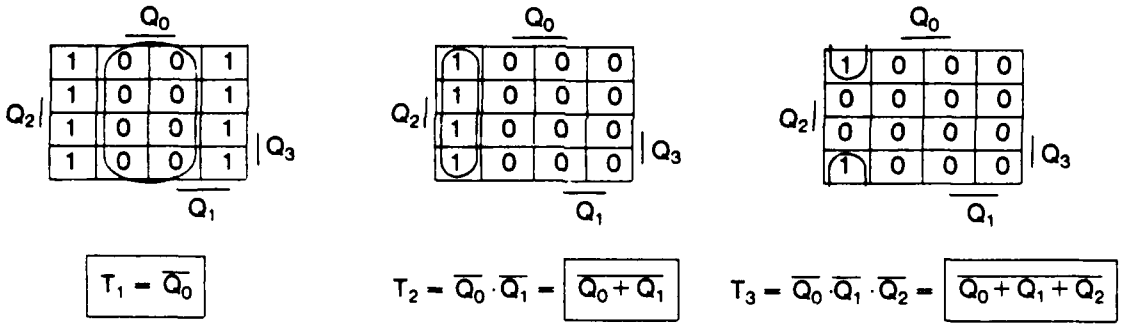
Figure 7.7. Divide-by-3 Counter

Divide-by-16 down-counter

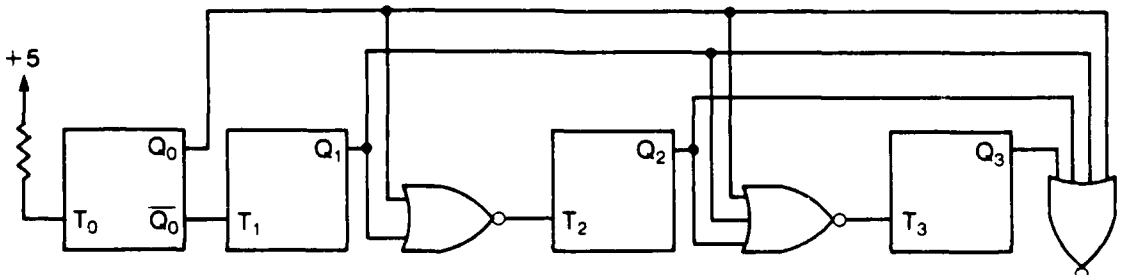
TRANSITION TABLE:

Input				Present state				Next state			
T_3	T_2	T_1	T_0	Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0
0	0	0	1	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	0	1	1	0	1
0	0	0	1	1	1	0	1	1	1	0	0
0	1	1	1	1	1	0	0	1	0	1	1
0	0	0	1	1	0	1	1	1	0	1	0
0	0	1	1	1	0	1	0	1	0	0	1
0	0	0	1	1	0	0	1	1	0	0	0
1	1	1	1	1	0	0	0	0	1	1	1
0	0	0	1	0	1	1	1	0	1	1	0
0	0	1	1	0	1	1	0	0	1	0	1
0	0	0	1	0	1	0	1	0	1	0	0
0	1	1	1	0	1	0	0	0	0	1	1
0	0	0	1	0	0	1	1	0	0	1	0
0	0	1	1	0	0	1	0	0	0	0	1
0	0	0	1	0	0	0	1	0	0	0	0
1	1	1	1	0	0	0	0	1	1	1	1

BY INSPECTION, $\boxed{T_0 = 1}$



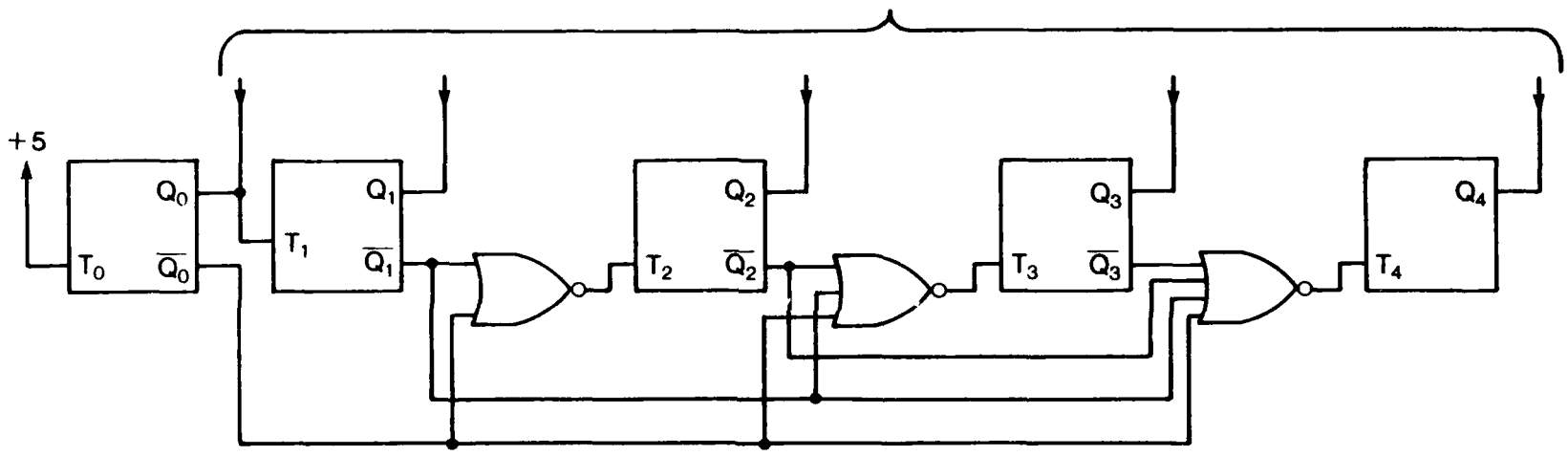
$\boxed{T_1 = \overline{Q_0}}$ $\quad T_2 = \overline{Q_0} \cdot \overline{Q_1} = \boxed{\overline{Q_0 + Q_1}}$ $\quad T_3 = \overline{Q_0} \cdot \overline{Q_1} \cdot \overline{Q_2} = \boxed{\overline{Q_0 + Q_1 + Q_2}}$



ALL CELLS ALSO GET:
SHIFT, $\phi_1 \phi_2$, LOAD, $\overline{\text{LD/SHIFT}}$

Figure 7.8. Divide-by-16 Counter

PROGRAM COUNTER - BLOCK DIAGRAM



ALSO,
ALL CELLS GET: SHIFT
ϕ_1
ϕ_2
$\overline{\text{LD/SHIFT}}$
LOAD

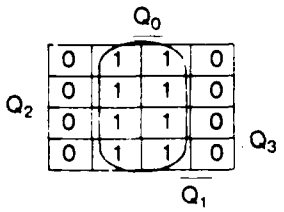
Figure 7.9. Program Counter

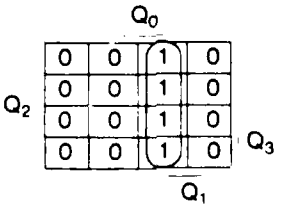
Program counter

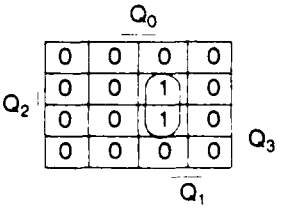
TRANSITION TABLE:

Input					Present state					Next state				
T_4	T_3	T_2	T_1	T_0	Q_4	Q_3	Q_2	Q_1	Q_0	Q_4	Q_3	Q_2	Q_1	Q_0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	1	0	0	0	0	1	1
0	0	1	1	1	0	0	0	1	1	0	0	1	0	0
0	0	0	0	1	0	0	1	0	0	0	0	1	0	1
0	0	0	1	1	0	0	1	0	1	0	0	1	1	0
0	0	0	0	1	0	0	1	1	0	0	0	1	1	1
0	1	1	1	1	0	0	1	1	1	0	1	0	0	0
0	0	0	0	1	0	1	0	0	0	0	1	0	0	1
0	0	0	1	1	0	1	0	0	1	0	1	0	1	0
0	0	0	0	1	0	1	0	1	0	0	1	0	1	1
0	0	1	1	1	0	1	0	1	1	0	1	1	0	0
0	0	0	0	1	0	1	1	0	0	0	1	1	0	1
0	0	0	1	1	0	1	1	0	1	0	1	1	1	0
0	0	0	0	1	0	1	1	1	0	0	1	1	1	1
1	1	1	1	1	0	1	1	1	1	1	0	0	0	0
0	0	0	0	1	1	0	0	0	0	1	0	0	0	1
0	0	0	1	1	1	0	0	0	1	1	0	0	1	0
0	0	0	0	1	1	0	0	1	0	1	0	0	1	1
0	0	1	1	1	1	0	0	1	1	1	0	1	0	0
0	0	0	0	1	1	0	1	0	0	1	0	1	0	1
0	0	0	1	1	1	0	1	0	1	1	0	1	1	0
0	0	0	0	1	1	0	1	1	0	1	0	1	1	1
0	1	1	1	1	1	0	1	1	1	1	1	0	0	0
0	0	0	0	1	1	1	0	0	0	1	1	0	0	1
0	0	0	1	1	1	1	0	0	1	1	1	0	1	0
0	0	0	0	1	1	1	0	1	0	1	1	0	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	0	0
0	0	0	0	1	1	1	1	0	0	1	1	1	0	1
0	0	0	1	1	1	1	1	0	1	1	1	1	1	0
0	0	0	0	1	1	1	1	1	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	0	0	0	0	0

BY INSPECTION, $\boxed{T_0 = 1}$







$\boxed{T_1 = Q_0}$ $\quad T_2 = Q_0 \cdot Q_1 = \boxed{\overline{\overline{Q_0} + \overline{Q_1}}}$ $\quad T_3 = Q_0 \cdot Q_1 \cdot Q_2 = \boxed{\overline{\overline{Q_0} + \overline{Q_1} + \overline{Q_2}}}$

$$\boxed{T_4 = \overline{Q_0} + \overline{Q_1} + \overline{Q_2} + \overline{Q_3}}$$

Figure 7.9. Program Counter (Continued)

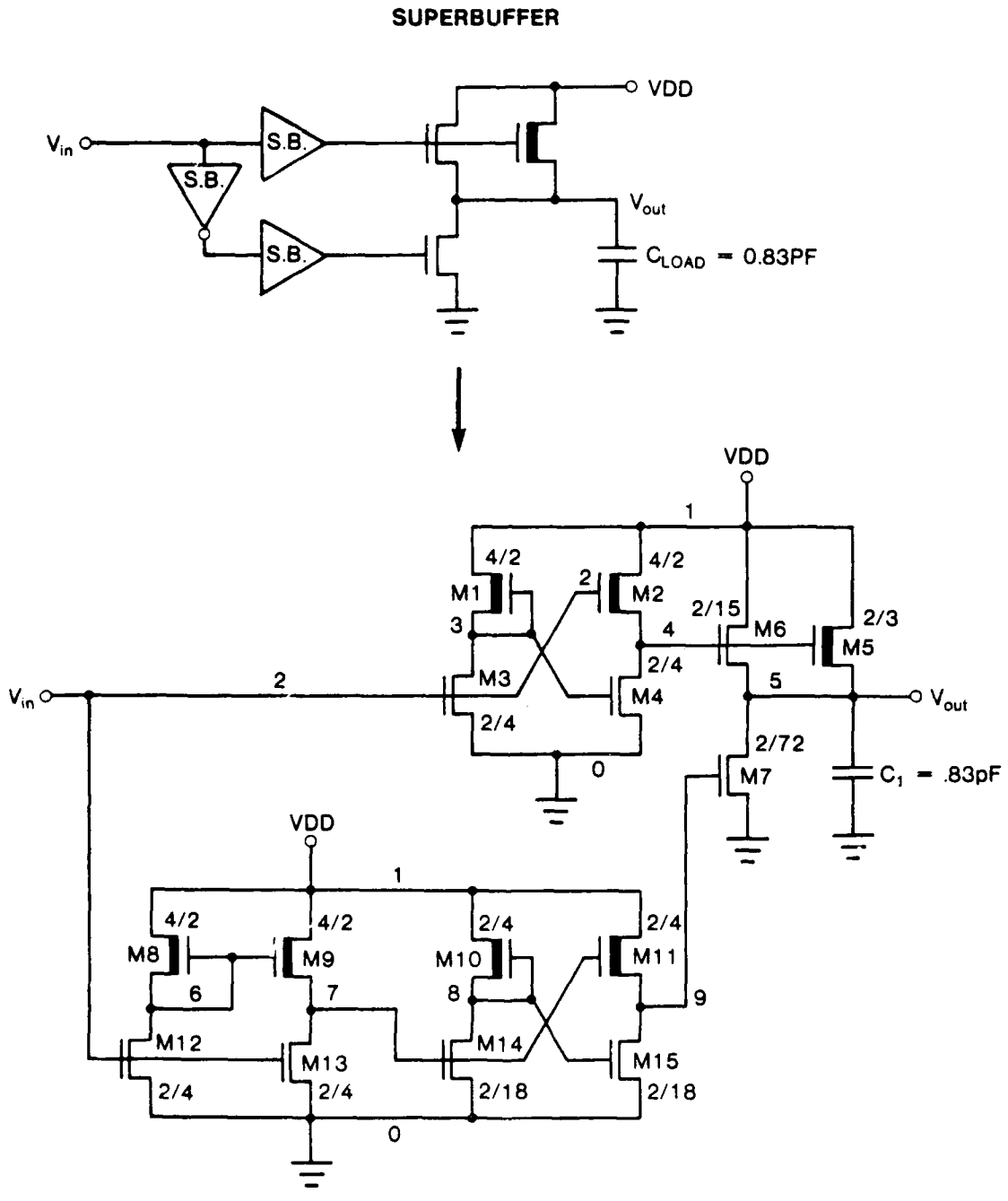


Figure 7.10. Superbuffers

Our design is a sequential design controlled by a two-phase non-overlapping clock. As long as the delay through the maximum number of levels is less than half the clock period, our design meets the level-sensitive requirement. For 10 Mhz operation, this time is 50 ns. Using SPICE, the longest path is simulated to determine maximum clock frequency.

The models for SPICE simulation of the program counter and the instruction PLA are shown in Figure 7.11,12,13. These two cells are the dominant blocks in terms of the speed limitations within the controller. The results are stated here:

Program counter

Propagation Delay-- 39 ns
Settling Time--22 ns

Instruction PLA

Propagation Delay-- 30 ns
Settling Time--45 ns

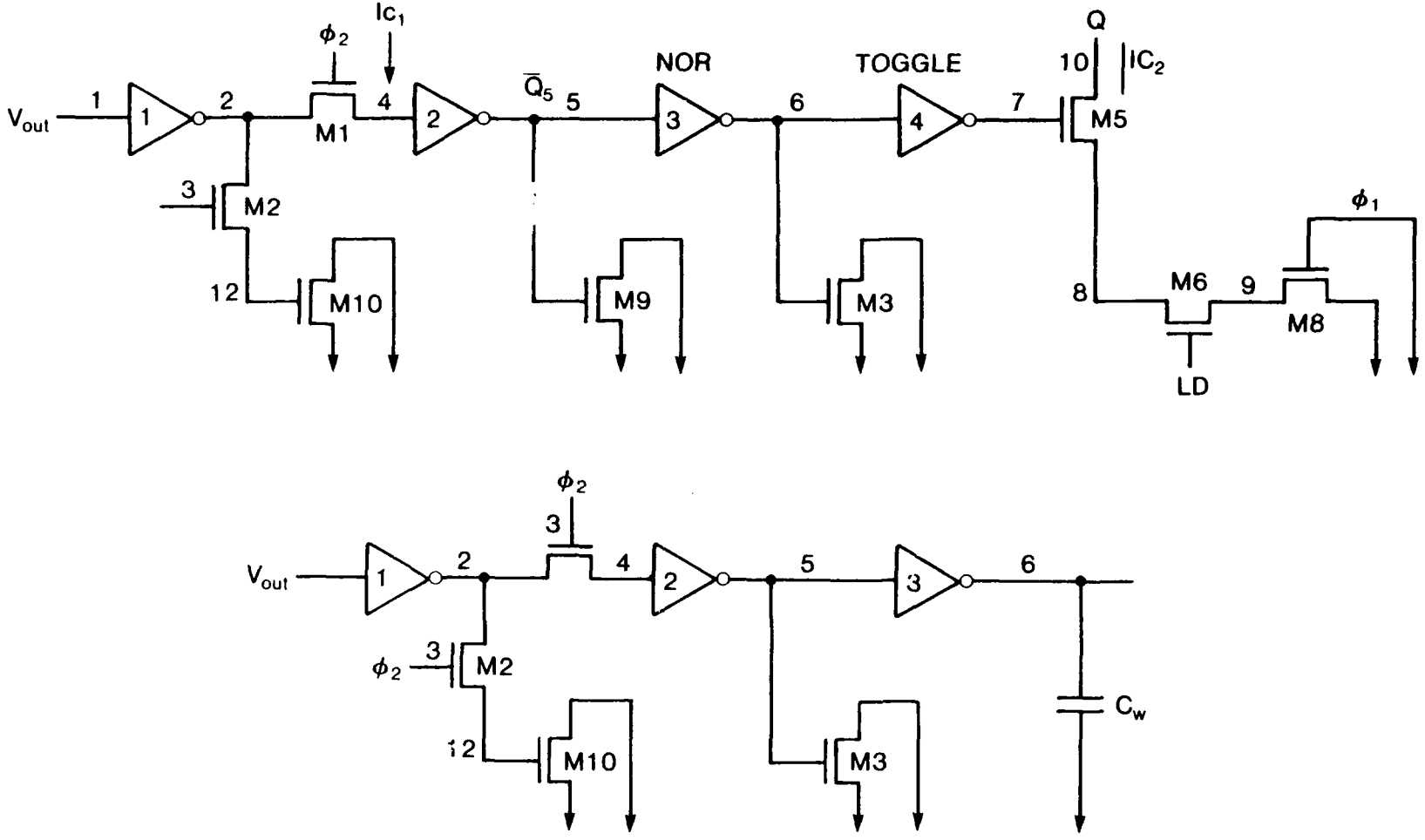


Figure 7.11. Circuit Model for Delay Calculation for the Program Counter

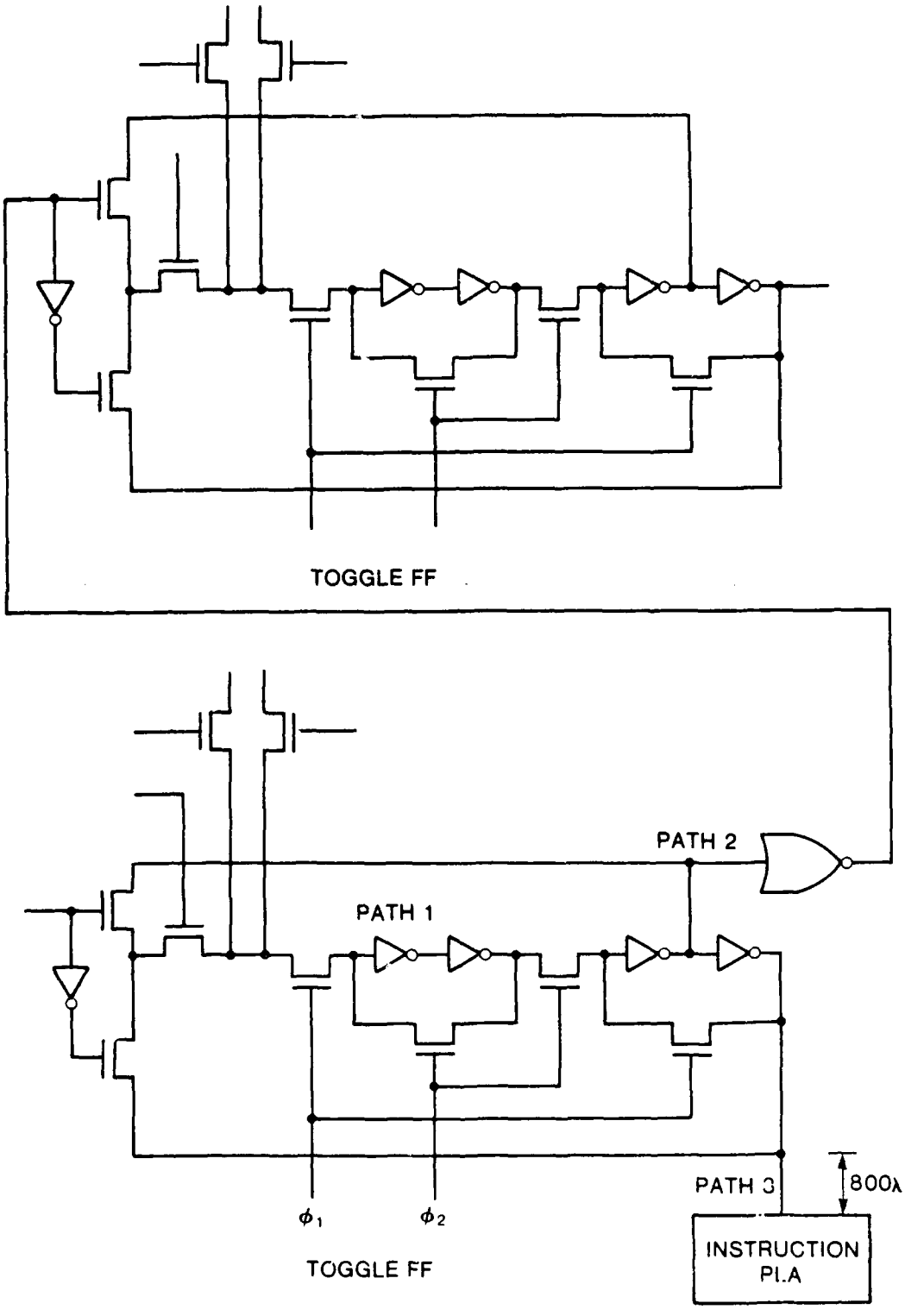
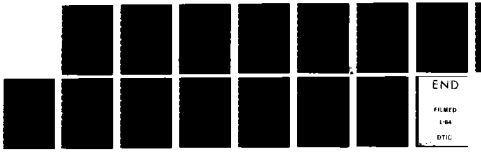
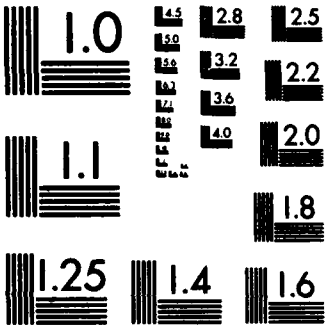


Figure 7.11. Circuit Model for Delay Calculation for the Program Counter

AD-A135 912 UCLA DEMODULATION ENGINE(U) CALIFORNIA UNIV LOS ANGELES 2/2
DEPT OF COMPUTER SCIENCE R SADR MAR 83 UCLA-ENG-83-23
MDA903-82-C-0064

UNCLASSIFIED F/G 9/5 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

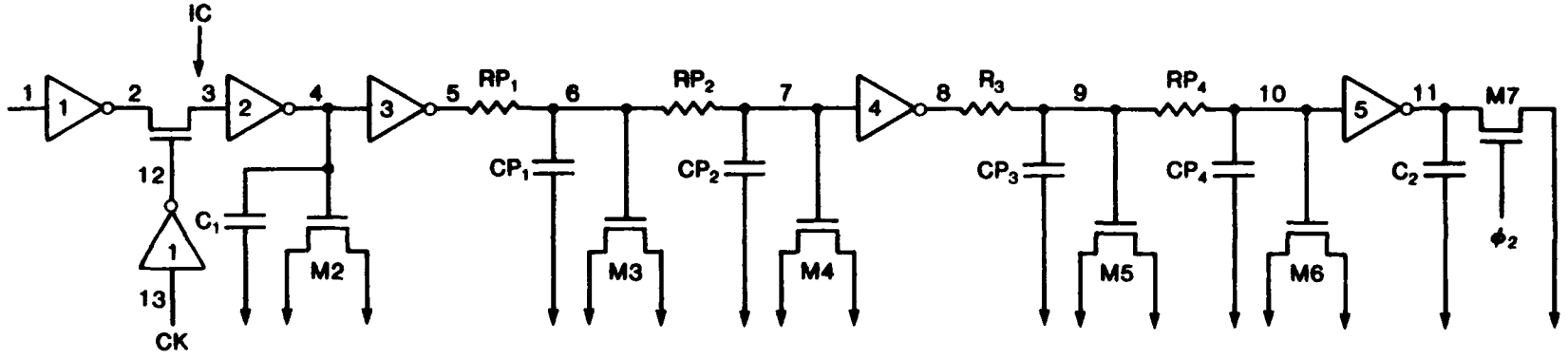


Figure 7.12. Circuit Model for Delay Calculation for the Instruction PLA

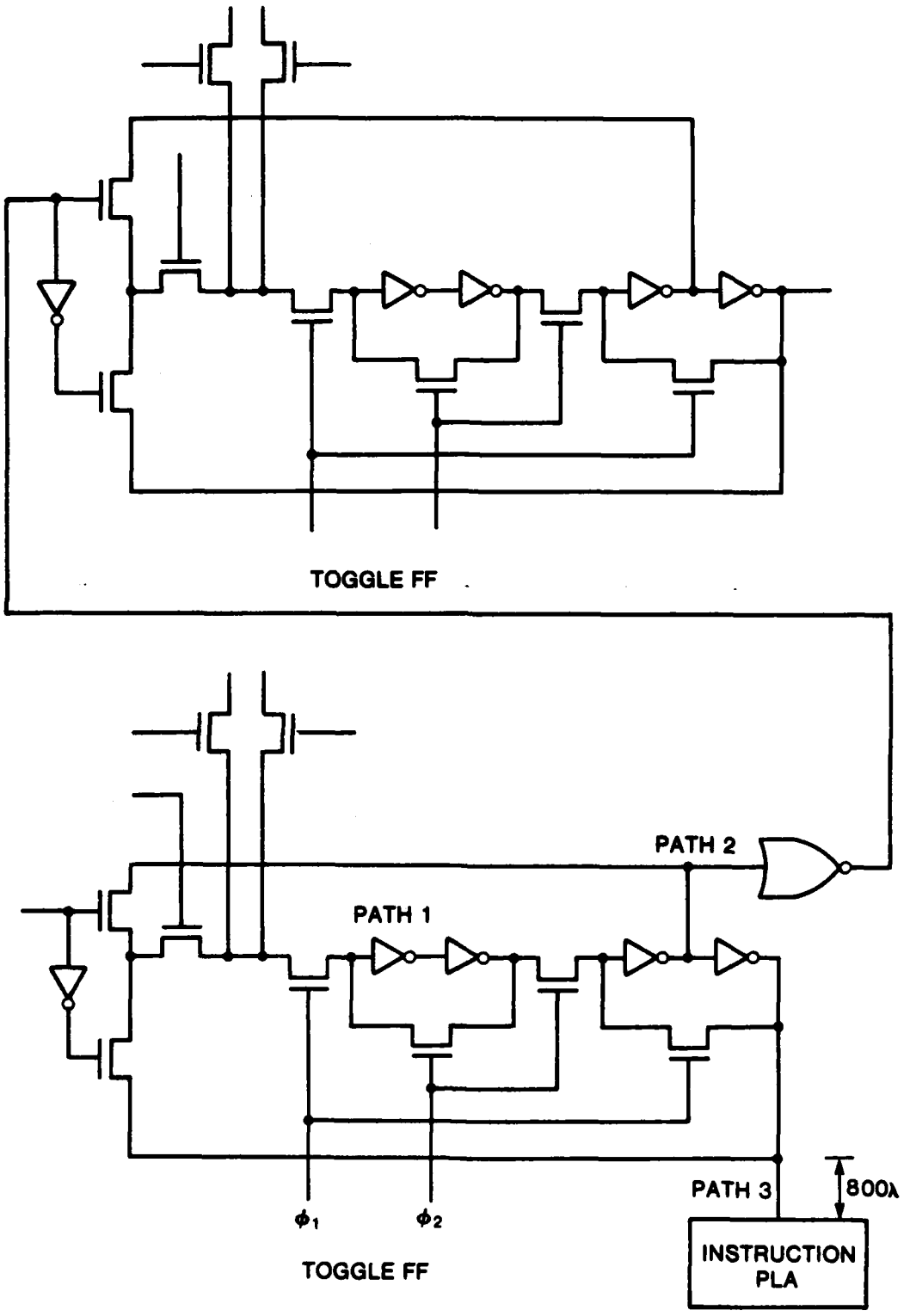


Figure 7.11. Circuit Model for Delay Calculation for the Program Counter

APPENDIX A
Theoretical Derivation of the Receiver

A.1 MSK

MSK belongs to a larger class of modulation techniques called Continuous Phase Modulations (CPM) which are also referred to as bandwidth efficient modulations [7,8,9] because of their superior spectral characteristics. The CPM signals use the constant envelope Frequency Modulated (FM) signal to transmit digital data in Radio Frequency (RF). The transmitted signal can be written as

$$X(t;\underline{a}) = \sqrt{\frac{2E_s}{T_s}}\, Cos\left[\omega_c t + \sum_{k=0}^{\infty} a_k h_k \pi \int_0^t g(\tau - kT_s)\, d\tau\right]$$

where

E_s is the energy of the transmitted signal
T_s is the symbol duration time in Seconds
a_k is the data symbol $\in \{\pm 1, \pm 3, \pm, \ldots, \pm M\}$
f_c is the carrier frequency in Hertz $f_c = 2\pi\omega_c$
$g(t)$ is the premodulation filter
h_k is the modulation index

MSK is the simplest form of the CPM signals with

(binary data) $a_k \in \{+1,-1\}$ $[0\rightarrow -1,\ 1\rightarrow +1]$

$h_k = \frac{1}{2}$ for *all* k

$$g(t) = \begin{cases} \frac{1}{2}T_s & 0 \leq t < T_s \\ 0 & T_s \leq t \end{cases}$$

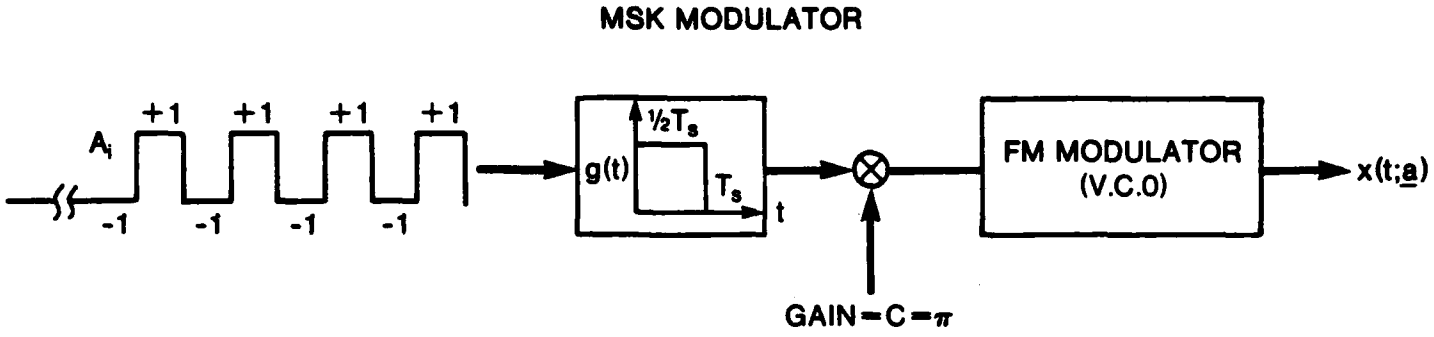
Figure A.1 illustrates the MSK modulator. The output of the premodulation filter $g(t)$ is

$$\int_0^t g(\tau)\, d\tau = \int_0^{t-nT_s} g(\tau)\, d\tau = \begin{cases} 0 & t \leq nT_s \\ \dfrac{t-nT_s}{2T_s} & nT_s \leq t < (n+1)T_s \\ \frac{1}{2} & (n+1)T_s \leq t \end{cases}$$

Hence, for the interval $nT_s \leq t < (n+1)T_s$, the MSK transmitted signal is

$$x(t;\underline{a}) = \sqrt{\frac{2E_s}{T_s}}\, Cos\left[\omega_c t + \frac{\frac{\pi}{2}a_n(t-nT_s)}{T_s} + \sum_{i=0}^{n-1}\frac{\pi}{2}a_i\right]$$

We define the "state" at the n^{th} time interval as



$$x(t;\underline{a}) = \sqrt{\frac{2E_s}{T_s}}\,\mathrm{Cos}\left(w_c t + a_n \frac{\pi}{2}\frac{(t-nT_s)}{T_s}\frac{\pi}{2} + \sum_{i=-\infty}^{n-1} a_i \frac{\pi}{2}\right)$$

$$t \in [nT_s, (n+1)T_s)$$

Figure A.1. MSK MODULATOR

$$S_n = \sum_{i=0}^{n-1} \frac{\pi}{2} a_i \quad Modulo-2\pi$$

The discrete process, S_n, can only take on four possible values since $a_i \in \{+1,-1\}$. Therefore, the "phase" space for MSK is

$$\Phi = \{0, \frac{\pi}{2}, 3\frac{\pi}{2}, \pi\}$$

The "state transition equation" in this case is

$$S_n = S_{n-1} + \frac{\pi}{2} a_{n-1} \quad Modulo-2\pi \quad S_n \in \Phi \tag{A.1}$$

Using (A.1), the state transition equation can be modeled by "the state diagram" as shown in Figure A.2.

A.1.2 MSK with Random Phase

In practice because of a lack of an ideal phase reference the MSK transmitted signal is

$$x(t;\underline{a},\phi) = \sqrt{\frac{2E_s}{T_s}} Cos\left[\omega_c t + \frac{\pi}{2} a_n \frac{(t-nT_s)}{T_s} + \frac{\pi}{2} \sum_{i=-\infty}^{n-1} a_i + \phi(t)\right]$$

where $\phi(t)$ is a stochastic process around the unit circle, this process has been studied previously [8,14,15] and in practice this process is usually a slowly varying random process. Therefore, it is assumed that the phase process remains constant during each $t \in [iT_s,(i+1)T_s)$, $i=0,1,2 \cdots$ *interval.*

In our proposed approach the phase space $[0,2\pi)$ is quantized into Q equal spaced intervals, as shown in Figure A.3.

$$\Phi = \{0,\Delta,2\Delta, \ldots, (Q-1)\Delta\} \text{ where } \Delta = \frac{2\pi}{Q}$$

Hence the process $\{\phi(t)\}$ can be approximated by the discrete Markov chain $\{\theta_i\}$ where

$$\theta_i = \theta_{i-1} + \phi_i \quad Modulo-2\pi \quad \theta_i \in \Phi$$

and ϕ_i is an independent identically distributed (i.i.d) sequence.

The Markov chain model θ_i $i = 0,1,2,...$ is essentially a quantized approximation to the true phase process $\phi(t)$ and with enough quantization values it can be made as accurate as necessary. Generally, channel noise will be the dominant source of degradation and, beyond a certain level of quantization, finer quantization would not affect the overall performance.

$$S_n = S_{n-1} + a_{n-1}\frac{\pi}{2}$$

$$a_n \in \{+1, -1\}$$

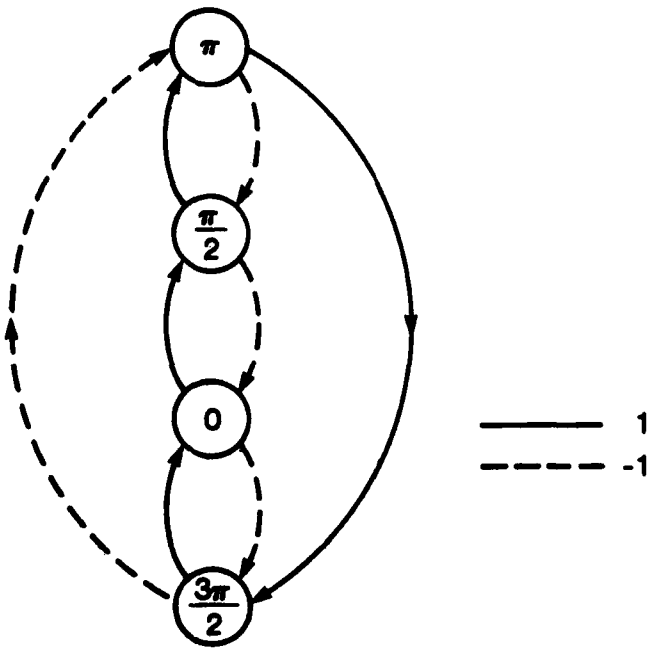


Figure A.2. MSK State Diagram

The quantized phase space for Q=16 is shown in Figure A.3. Φ constitutes the state space of the first order Markov chain S_n at time nT_s. For MSK with random phase, the state at time nT_s is

$$S_n = \sum_{i=0}^{n-1} \left(\frac{\pi}{2} a_i + \phi_i\right) \quad Modulo-2\pi \; ; S_n \in \Phi$$

The state transition equation is

$$S_n = S_{n-1} + \frac{\pi}{2} a_{n-1} + \phi_{n-1} \; Modulo-2\pi$$

where

$$S_n \in \Phi \quad a_n \in \{+1,-1\} \quad \phi_n \in \{-\Delta,0,\Delta\}$$

The assumption that $\phi_i \in \{-\Delta,0,\Delta\}$ means that the random phase process $\theta_i(Modulo-2\pi)$ jumps only to adjacent quantized phase values or it remains unchanged during a symbol interval T_s seconds. A typical state transition for Q=16 is shown in Figure 2.3.

A.2 Receiver Design

The digital communication problem is depicted in Figure A.4. The transmitted signal $x(t;\underline{a})$ is contaminated with Additive White Gaussian Noise (AWGN) with double sided flat spectral density No/2. The receiver's goal is to find the best estimate of the data using the observed signal $y(t) = x(t;\underline{a}) + n(t)$.

We shall use "soft decision" [1] decoding which reduces the bit error probability as opposed to "hard decision" [1]. The simultaneous phase tracking and data demodulation of the CPM signals was studied by Jackson [8].

To find the optimum receiver, the Maximum A-posteriori estimation rule (MAP) is used[1]. This estimation rule selects the sequence pair $(\hat{\underline{a}},\hat{\underline{\phi}})$ that maximizes the joint probability of the data and the phase, given the observed vector $\underline{y}$. That is,

$$(\hat{\underline{a}},\hat{\underline{\phi}}) = \underset{(\underline{a},\underline{\phi})}{Max^{-1}} P(\underline{a},\underline{\phi}|\underline{y})$$

$$\text{where } \underline{a} = (a_1, \ldots, a_k) \quad \underline{\phi} = (\phi_1, \ldots, \phi_k) \quad \underline{y} = (y_1, \ldots, y_L) \; [2]$$

The phase and the data are independent processes and $p(\underline{y})$ does not affect the maximization, therefore, the maximization is equivalent to

[1] This rule minimizes the probability of error [1].

[2] The Gram-Shmidt orthognalization procedure can be used to find this vector.

$$\{\phi(t)\} \leftrightarrow \{\theta_n = \theta_{n-1} + \phi_{n-1}\} \qquad \phi_n \in \{-\Delta, 0, \Delta\}$$

$$\Phi = \{0\Delta, 2\Delta, \ldots, (Q-1)\Delta\} \quad \Delta = 2\frac{\Pi}{Q}$$

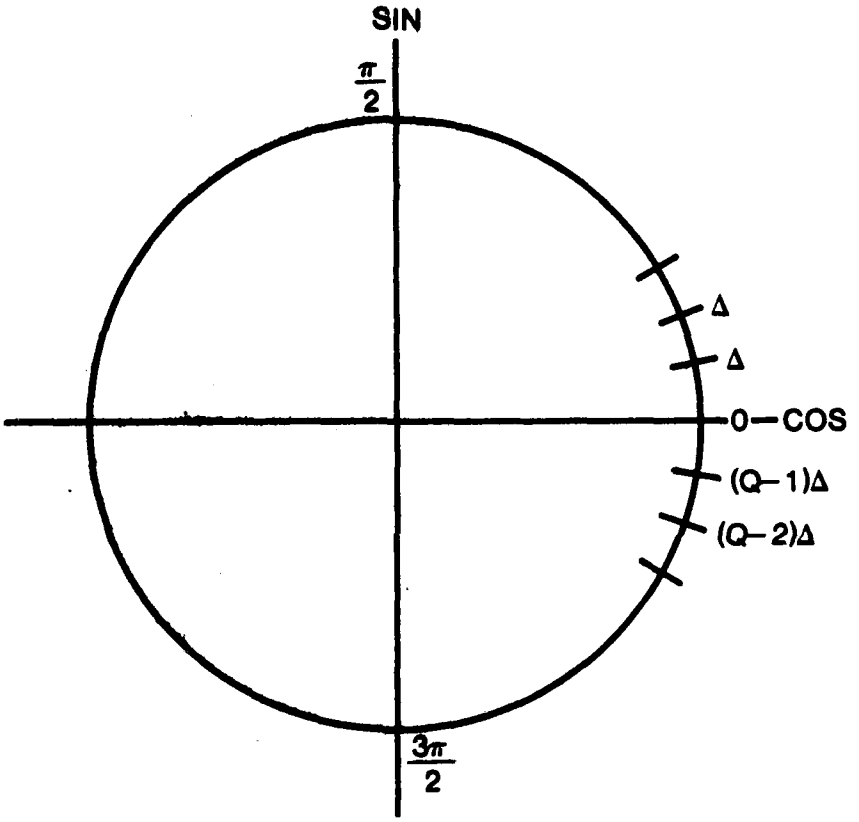


Figure A.3. Phase Process

$$\underset{(\underline{a},\underline{\phi})}{Max}\, P(\underline{y}|\underline{a},\underline{\phi})P(\underline{a})P\underline{\phi})$$

Furthermore, a_n is assumed to belong to the equiprobable alphabet $U=\{+1,-1\}$, $P(a_i=1) = P(a_i=-1) = \frac{1}{2}$, thus we have

$$\underset{(\underline{a},\underline{\phi})}{Max}\, P(\underline{y}|\underline{a},\underline{\phi})P(\underline{\phi})$$

Denoting natural logarithm as ln

$$\underset{(\underline{a},\underline{\phi})}{Max}\, \ln P(\underline{y}|\underline{a},\underline{\phi})P(\underline{\phi})$$

We shall further assume that the phase process is an independent process, so that,

$$\underset{(\underline{a},\underline{\phi})}{Max}\, \ln P(\underline{y}|\underline{a},\underline{\phi})P(\phi_1)\cdots P(\phi_k)$$

The maximization of the conditional probability density $P(\underline{y}|\underline{a},\underline{\phi})$, which is Gaussian in this case, is equivalent to maximizing the $L_{2[0,kT_s]}$ (Hilbert space) inner product of the transmitted signal $x(t;(\underline{a},\underline{\phi}))$ with the observed signal y(t). This rule is given by,

$$\underset{(\underline{a},\underline{\phi})}{Max} \int_0^{kT_s} x(t;(\underline{a},\underline{\phi}))y(t)\,dt - \sum_0^k \frac{No}{2}\ln P(\phi_i)$$

Defining a few terms to simplify the above expression, y(t) is projected along the in-phase and quadrature components of x(t) given by

$$y_{n,c}(a_n) = \int_{nT_s}^{(n+1)T_s} \sqrt{\frac{2E_s}{T_s}}\, Cos\left(\omega_c t + \frac{\pi}{2}a_n\frac{(t-nT_s)}{T_s}\right)y(t)\,dt$$

$$y_{n,s}(a_n) = -\int_{nT_s}^{(n+1)T_s} \sqrt{\frac{2E_s}{T_s}}\, Sin\left(\omega_c t + \frac{\pi}{2}a_n\frac{(t-nT_s)}{T_s}\right)y(t)\,dt$$

Noting the expansions

$$\int_0^{kT_s} x(t;(\underline{a},\underline{\phi}))y(t)\,dt = \sum_{i=0}^{k-1}\int_{iT_s}^{(i+1)T_s} x(t;(\underline{a},\underline{\phi}))y(t)\,dt$$

$$\text{for } nT_s \leq t < (n+1)T_s \quad x(t;(\underline{a},\underline{\phi}) = \sqrt{\frac{2E_s}{T_s}}\, Cos\left(\omega_c t + \frac{\pi}{2}a_n\frac{t-nT_s}{T_s} + \sum_{i=0}^{n-1}(\frac{\pi}{2}a_i+\phi_i)+\phi_n\right)$$

$$-\sqrt{\frac{2E_s}{T_s}}\left(Cos(\omega_c t + \frac{\pi}{2}a_n\frac{(t-nT_s)}{T_s})\,Cos(S_n+\phi_n) - Sin(\omega_c t + \frac{\pi}{2}a_n\frac{(t-nT_s)}{T_s})\,Sin(S_n+\phi_n)\right)$$

where

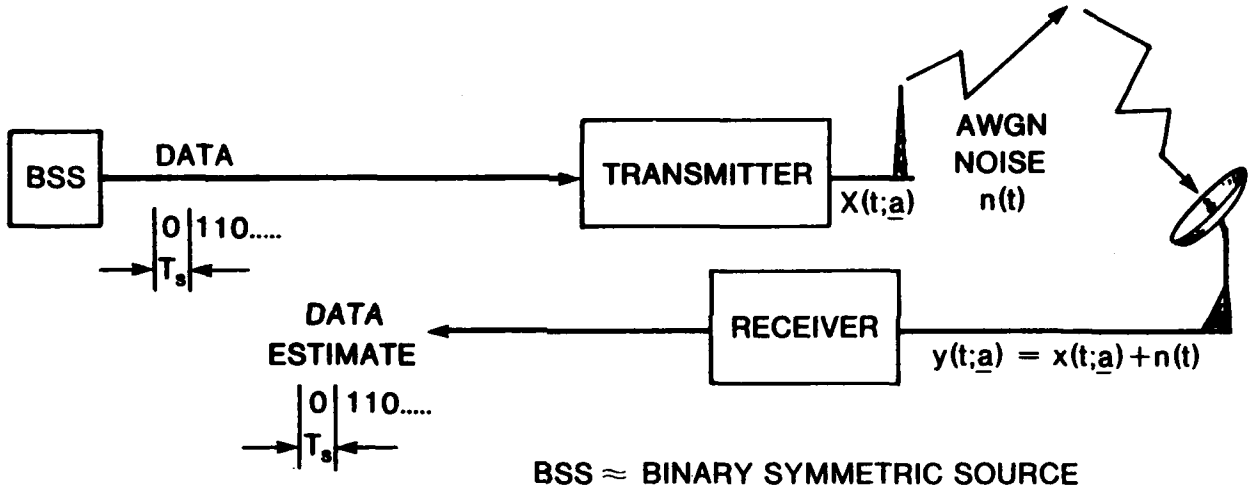


Figure A.4. Digital Communication Problem

$$S_n = \sum_{i=0}^{n-1} (\frac{\pi}{2} a_i + \phi_i) \quad Modulo-2\pi$$

or

$$S_n = S_{n-1} + \frac{\pi}{2} a_{n-1} + \phi_{n-1} \quad Modulo-2\pi$$

define the "branch metric"

$$m(\underline{y}_n; a_n, \phi_n, S_n) \triangleq \int_{nT_s}^{(n+1)T_s} y(t) x(t; (\underline{a}, \underline{\phi}) dt = y_{n,c}(a_n) Cos(S_n + \phi_n) + y_{n,s}(a_n) Sin(S_n + \phi_n)$$

where $\underline{y}_n \in R_4$ (4-dimensional Euclidean Space).

$$\underline{y}_n = (y_{n,c}(1), y_{n,c}(-1), y_{ns}(1), y_{ns}(-1))$$

This maximization problem is reduced to

$$\max_{(\underline{a}, \underline{\phi})} \sum_{i=0}^{k} m(\underline{y}_i; a_i, \phi_i, S_i) - \frac{No}{2} \ln P(\phi_i)$$

$$S_i = S_{i-1} + \frac{\pi}{2} a_{i-1} + \phi_{i-1}; S_i \in \Phi$$

$$\underline{a} = (a_0, \ldots, a_k) \; a_i \in \{+1, -1\}$$

and we shall further assume that the phase process $\phi_i \in \{\Delta, 0, -\Delta\}$, i.e., the phase value jumps only to adjacent quantized phase values. If these transitions are equally likely then $P(\phi_i = 0) = P(\phi_i = \Delta) = P(\phi_i = -\Delta) = 1/3$, and thus

$$\max_{(\underline{a}, \underline{\phi})} \sum_{i=0}^{k} m(\underline{y}_i; a_i, \phi_i, S_i)$$

The Viterbi algorithm [3,16] is the dynamic programming solution to the above maximization problem. The state transition equation $S_i = S_{i-1} + \frac{\pi}{2} a_{i-1} + \phi_{i-1}$ can be represented in the form of a state diagram; the time sequence presentation of the state diagram for i=0,1,2 , . . . , k is called the trellis diagram (Figure 2.2.a).

For every transition defined by the state transition equation, there is a metric associated with this transition, which we shall refer to as the branch metric value and denote as $m(S_n; S_{n+1})$, for notational convenience. Here this is

$$m(S_n; S_{n+1}) = y_{n,c}(a_n) Cos(S_n + \phi_n) + y_{n,s}(a_n) Sin(S_n + \phi_n)$$

where

$$S_{n+1} = S_n + \frac{\pi}{2} a_n + \phi_n$$

The key point in applying the Viterbi algorithm is the forward equation for the value of the accumulated metric value of every state of the trellis diagram given by

$$AccMet(S_n) \triangleq \underset{S_n}{Max} \left[AccMet(S_n) + m(S_n;S_{n+1}) \right]$$

$$\textit{with initial condition} \quad AccMet(S_0) = 0 \quad n = 0,1,2, \ldots, k$$

A.3 Summary

In summary, the mathematical problem is to find $(\hat{\underline{a}},\hat{\underline{\phi}})$ such that

$$\underset{(\underline{a},\underline{\phi})}{Max} \sum_{i=0}^{\infty} m(S_i;S_{i+1})$$

where

$$m(S_i;S_{i+1}) = y_{i,c}(a_i) Cos(S_i + \phi_i) + y_{i,s}(a_i) Cos(S_i + \phi_i)$$

$$S_{i+1} = S_i + a_i \frac{\pi}{2} + \phi_i$$

The solution is given by dynamic programming also known as the Viterbi algorithm when there are finite number of states. The transformation of this formulation of the optimum demodulator into a VLSI chip is contained in Chapter 2.

APPENDIX B
MICROCODES

"Symbol Cycle"

CPU | TPU | MEMORY

```
T0: Ycs(1),Ycs(-1);accept new input values| NOP |Dout<--D0;output decoded bit
    Coef1<-----X1  Coef2<-----X2      |
T1: Call <<Multiplier Routine >>| Call <<TPU Queue Routine>> | NOP
  ;This call will result in having the first state I=0 Ls1,ls2,Ls3
  ;in the TPU queue;in the meantime we shall compute bm(1) for I=0.


        DO T8 I=0,15


        DO T3 j=1,3

T2: Call<< Branch metric Routine>> & <<Multiplier Routine>>
  ;this call is for "0" transitions and the second routine is called
  ;for to compute bm(1) concurrently.Note here that the second call
  ;can be extended over the complete cycle of this loop, i.e., while
  ;the survivor for "0"transition is found we are also computing
  ;the branch metric for transition "1".

T3: Call <<Survivor routine>>        |  NOP  |  NOP
  ;this call is to find the survivor ,again  this loop is for "1"

T4:          NOP              |It<------It+1|    NOP
T5:          NOP         |Coef1<----X1  Coef2<---X2| NOP
  ;these two cycles causes the TPU to point to the next state so
  ;that in the next three cycles the multiplier can compute the
  ; bm(0) for the I=I+1 state.


        DO T5 j=1,3

T4:Call <<Branch metric Routine>>&<<Multiplier Routine>>
  ;this call is for only "1" transitions,note the comment during
  ;T5.

T5:Call  <<Survivor Routine >>       |  NOP  |  NOP
```

```
;At this point we have the so called "survivor" of state I.
;for the corresponding figure refer to system specification
;document page().

T6: XN<----Y   LSS<----Yad    | NOP | Accmetout<-----Y  Din<-----Ds
;The survivor's absolute value and it's corresponding state
;output via the memory.In the cpu itself we store the survivor's
;absolute value in the normalization input register to be compared
; with other 16 survivors.

T7:Call <<Comparator Normalization Routine>>| NOP | PA(I)<---BP(LSS)
                                            | BAM(I)<--Accmetout
;CPU compares the survivor accumulated metric to its previous value
;and stores normalization constant.The memory stores the path"properly"
;and the state I's accumulated metric.

T8:Clear            |   NOP     |  PA(I)<---shift in---Din
;All CPU registers are Cleared EXEPT YN,to repeat the process for the
;next state .The decoded bit is shifted into the path memory.

T9:  N<----YN       |    NOP    |    NOP
;this cycle is the initialization for the next symbol time operation
;so the normalization constant is clocked into the normalization
;registers ,which is to be used in the <<Branch Routine>>

T10:  YN<----Fill with "1"s | I<----0   | Call <<Replicate Routine>>
;the normalization register output is filed with "1"s so that
;in the following cycles the smallest value can be found.
;Memory 's background and foreground registers are replicated.


   List of Subroutines:


 <<Multiplier Routine>>

t1: bm<-----X1*Y1+X2*Y2+7       | NOP   |   NOP
;This routine has many other submicroinstruction which will be
;appended.its function is to compute the branch metric values

 <<Branch Routine>>
t1:      NOP       | LS<-----Lsq1    | NOP
;TPU provides the last state j of present state I

t2:      NOP       | Lsq3<----Lsj+3(It)  | Read AM(LS)
;The accumulated metric value of the ls state is read from
;the memory.The queue moves down and a fresh value is input
;to the queue.

t3: A<----AM(LS)-N      | Dout"1"or "0" |      NOP
;The accumulated metric is normalized and TPU outputs
```

```
   ;decoded bit via the survivor comparator.

t4: X<----A+bm   Xad<---LS  DX<---Dout| NOP  |   NOP
   ;CPU computes the accumulated metric and inputs it via
   ;the survivor comparator with its corresponding last state
   ;and decoded bit.

  <<Survivor Routine >>
t1: if X>Y then: Y<----X Yad<----Xad DS<-----D|   NOP |NOP
   ;Survivor comparator shifts in the value of x into y register
   ;if x>y


  <<Comparator Normalization Routine >>
t1: if XN<YN then YN<------XN              |  NOP  |NOP
   ;note here that at the end of every symbol cycle YN is
   ;filled with 1s.

  <<Replicate Routine >>
t1:  NOP            |  NOP | AM(I)<-----BAM(i) for all i=0,15
                         | PA(i)<-----BAP(i) for all i=0,15
   ;This will occur at the end of every symbol cycle to replicate
   ;the content of all the backups and front registers in the memory.

  <<TPU Queue Routine>>
t1:  NOP            | Lsq3<----Ls1(0)|   NOP
                   |Ls<----Lsq1 Ls<----Lsq1   |NOP
    ;this will fill the queue with the last states which lead to I=0,
    ; namely 3,4,5.
```

APPENDIX:

X2 ,X1 ~~coef stored in TPU
It~~index register in TPU
bm~~ branch metric register
AM(k)~~accumulated Metric Value corresponding to state k
PA(k)~~Path Memory FIFO corresponding to state k
D~~Decoded bit stored in TPU
DS~~Survived decoded bit register in CPU
LS~~Last state BUS from TPU
LS(j)~~Last state from j=1,6 in TPU
N~~Normalization Value Register in CPU
XN~~Normalization register input in CPU
YN~~Normalization register in CPU
Xad~~Survivor Comparator register input address in CPU
Yad~~ Survivor comparator register output address in CPU
Y~~ Survivor comparator register output absolute value in CPU
X~~ Survivor comparator register input absolute value in CPU

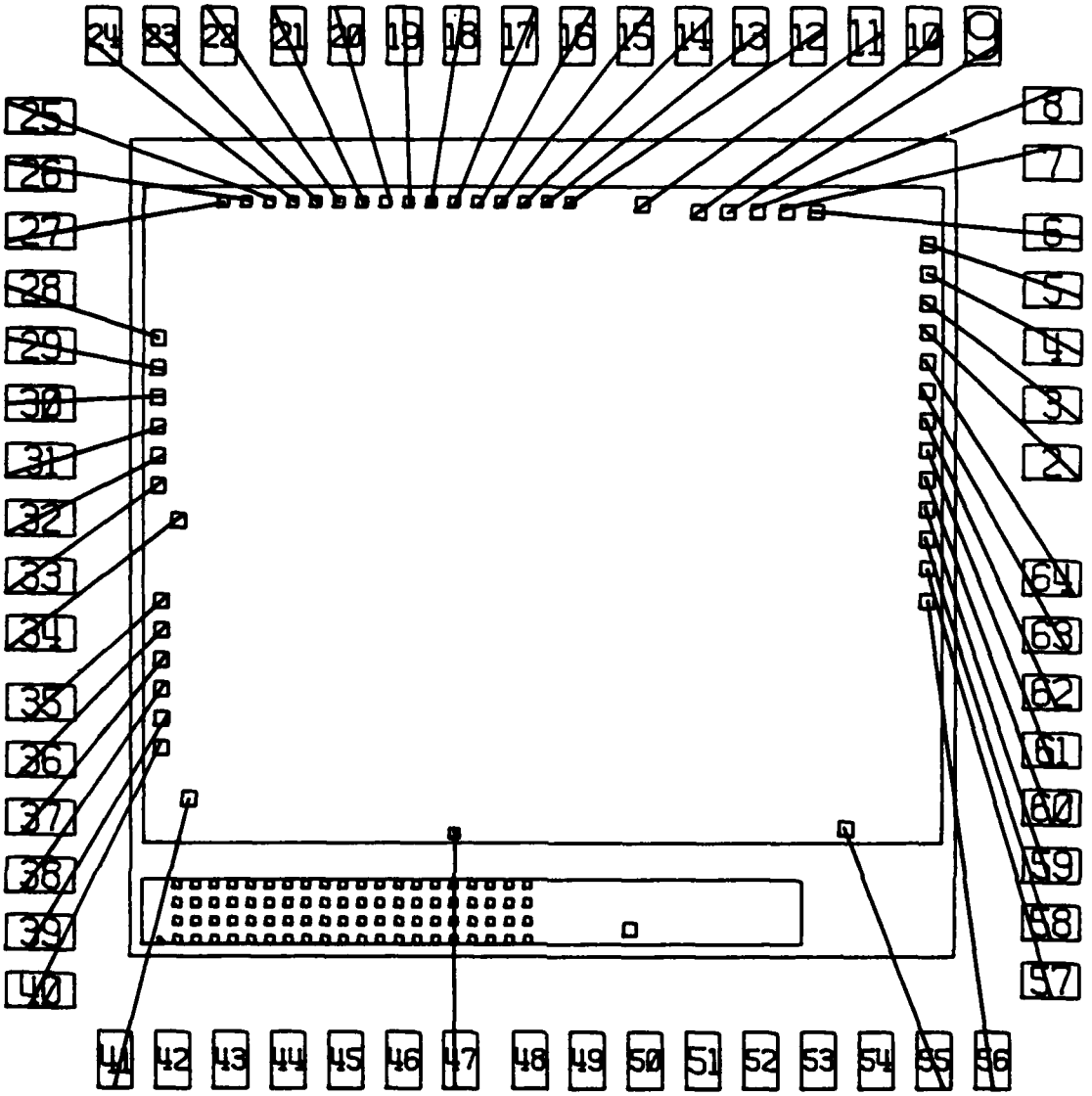
DS~~Survivor comparator register output bit in CPU
DX~~Survivor comparator register input bit in CPU
Din~~Survivor comparator register input bus from TPU
Coef12~~registers for X1 ,X2 cof in CPU multiplier

APPENDIX C

PIN ASSIGNMENT

Those I/O which are intended for level sensitive scan testing are abreviated as (LTST). Those I/O which are real valued, their ordering is deonted by MSB standing for the Most Significant Bit and LSB for the List Significant Bit.

1. Substrate VSS
2. Change input Yc,Ys
3. Test Data Out (LTST)
4. Test Data Input (LTST)
5. Reset (LTST)
6. Shift Signal (LTST)
7. Test Clock (LTST)
8. Main Clock
9. GRD
10. Ys(1) Sign
11. Ys(0) Sign
12. Yc(1) Sign
13. Yc(0) Sign
14. Ys(1) LSB
15. Ys(0) LSB
16. Ys(1)
17. Ys(0)
18. Ys(1) MSB
19. Ys(0) MSB
20. Yc(1) LSB
21. Yc(0) LSB
22. Yc(1)
23. Yc(0)
24. Yc(1) MSB
25. Ys(1) MSB
26. Normalizer Constant MSB
27. Normalizer Constant
28. Normalizer Constant
29. Normalizer Constant
30. Normalizer Constant
31. Normalizer Constant LSB
32. GRD
33. Last State MSB
34. Last State
35. Last State
36. Last State LSB
37. Unused
38. Unused
39. GRD
40. Unused
41. Unused
42. Unused
43. Unused
44. Unused
45. VDD
46. Unused
47. Unused
48. Unused
49. Unused
50. Unused
51 Unused
52. Unused
53. GRD
54. SYSTEM RESET
55. Dout (DATA via User)
56. I- Present State MSB
57. I- Present State
58. I- Present State
59. I- Present State LSB
60. Present Controller State P0 MSB
61. Present Controller State P0
62. Present Controller State P0
63. Present Controller State P0
64. Present Controller State P0 LSB



M29TC Ø

CLA

PKG64

References

1. Viterbi, A.J. and Omura, J.K. *Principles of Digital Communication and Coding,* McGraw Hill Company, 1979.

2. Mead, C. and Conway, L. *Introduction to VLSI System,* Addison and Wiley Company, 1980.

3. Omura, J.K. "On The Viterbi Algorithm," *IEEE,* Vol. IT-15, January 1969.

4. Forney, G.D. "The Viterbi Algorithm," *IEEE Proceedings,* Vol. 61, No. 3, March 1973.

5. Viterbi, A.J. *Principles of Coherent Communications,* McGraw Hill Company, 1962.

6. Anderson, R.R. and Salz, J. "Spectra of Digital FM," *Bell Systems Technical Journal,* Vol. 44, pp. 1165-1189, July-August 1965.

7. Aulin T. *CPM-A Power and Bandwidth Efficient Digital Constant Envelope Modulation Scheme,* Dr. Dissertation, Univ. Lund, Lund, Sweden, November 1979.

8. Jackson, D.E. *Bandwidth Efficient Modulation and Coding,* Ph.D. Dissertation, UCLA, 1980.

9. "Special Issue on: Combined Modulation and Encoding," *IEEE Transactions on Communications,* Vol. 29, No. 3, March 1981.

10. Bayer, J.L. *Computer Systems Architecture,* Computer Science Press, 1980.

11. Carr, W. and Mize, J. *Mos/VLSI Design and Application,* McGraw Hill Company, 1972.

12. Eichlberger E.B. and Williams T.W. "A Logic Design Structure for LSI Testability," *J. Des. Autom. and Fault Tolerant Comput.,* Vol. 2, No. 2, May 1978.

13. Mazur B.A. and Taylor D.P. "Demodulations and Carrier Synchronization of Multi-h Codes," *IEEE Transactions on Communications,* Vol. 29, No.3, March 1981.

14. Ungerbock, G. "Channel Coding with Multilevel/Phase Signals," *IEEE Information Th. Transactions,* Vol. 28, No. 1, January 1982.

15. Scharf, L.L., Cox, D.D. and Mazereliz, C.J. "Modulo-2 Phase Sequence Estimation," *IEEE Information Th. Transactions,* Vol. 6, No. 5, September 1980.

16. Macchi, O. and Scharff, L.L. "A Dynamic Programming Algorithm for Phase Estimation and Data Decoding on Random Phase Channel," *IEEE Information Th. Transactions,* Vol. 27, No. 5, September 1981.

END

FILMED

1-84

DTIC